August 2021

Framing Ludens: Pawn Swapping and Game Mode Alteration in an Unreal Engine Game Level

Jeffrey Paul Martell
Clemson University, jpmartell@gmail.com

Framing Ludens: Pawn Swapping and Game Mode Alteration in an Unreal Engine Game Level

---

A Dissertation
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Master of Fine Arts
Digital Production Arts

---

by
Jeffrey Paul Martell
August 2021

Accepted by:
Dr. Eric Patterson, Committee Chair
David Donar
Insun Kwon

Abstract

Knight of Drones is a hybrid twin-stick top-down shooter and side-scrolling platformer that aims to bring vintage gameplay to a contemporary game engine. A single swatch of the top-down game mode and a pair of levels from the side-scrolling game mode will be presented beginning with the player characters and player pawn assets and then extending to the level design and game design assets.

The game deals with the fallout effects of climate change and serves as a cautionary tale against pollution and weaponized AI. This message will appear primarily in the game's atmosphere, literally in the background in some cases, as the settings and places visited by the player will be constructed of level assets that relate to these concepts. Instead of standard platformer levels (the ice level, the lava level, the jungle level), Knight of Drones features levels such as an oil slick ocean level, an abandoned copper smelter level, or a plastic dump level.

While not blind to the irony of using silicon processing power to warn about the negative effects of consumer waste on the environment, there exists an undervalued opportunity to build game worlds that promote social causes. By creating a setting that engages the player through environmental instability, and by using familiar, approachable vintage mechanics, it may be possible to celebrate the history of gaming and offer the player thoughtful moral questions without diluting the core gameplay mechanics or taking agency away from the player.

Hybridity of game mode is used in Knight of Drones to change up the gameplay speed and style as it affords the player more than one viewpoint or character token to control. Additional hybridity of genre should offer the players something innovative in aesthetics. Viewed from the Mechanics Design Aesthetics (MDA) framework, the goal of Knight of Drones is to offer old-school gameplay in a strange new setting that makes the player consider humanity's long-term effects on the planet.

Dedication

This research and production is dedicated to my father Paul Martell. Thank you for guiding me, and always being there. May the SEGA never come between us.

Acknowledgments

I hold immense gratitude for the instruction and support of Eric Patterson, Jessica Baron, Insun Kwon, David Donar, and Victor Zordan. Additional thanks to the faculty, staff, and Visual Arts students at the South Carolina Governor's School for the Arts and Humanities. Your patience and flexibility made this work possible.

Artist Statement

This gameplay prototype is the result of my desire to learn how to design and implement a video game. It does not reflect a full game in its entirety, but rather a smaller compartment of learning and development of an artist seeking to bolster a set of technical abilities and practice contemporary forms of computer generated art asset creation. It's been a welcome challenge to balance artistic creation and technical execution, and one of the more important lessons along the way has been the dissolution of the barrier between the two. Artistic creation can be technical and studious, while technical implementation can be quite creative and flow-state inducing. The idea that the two are separate may be a false dichotomy perpetuated by specialization and team delineation; however, the arguments that persist seem to stem from a meaningful place.

As a young man I fell in love with Capcom's side-scrolling action games, particularly the MegaMan games, which allow the player to select the order in which they tackle the levels and bosses. To modern gamers, these are old school games with unforgiving jump puzzles and trial-and-error problem solving. To a young child new to the concept of a video entertainment system, this was a revelatory new world to explore. Growing up with the age of consoles and personal computers, I have witnessed the growth of video games from shunned, nerdy pastime into monstrous multi-billion dollar industry and popular culture maelstrom.

The formula for Knight of Drones is simple: old school gameplay with new school design. My dedication to story forms the foundation for the concept of a post-human world with only the scrap heaps of automated military machines fighting with one another over a forgotten programmatic purpose. My background in story art and filmmaking leads me to seek ideas that encourage thought and dialog among audience members. Science fiction is a wonderful genre with which to ask important questions about humanity's actions and its effects on our world. The fiction of Neal Stephenson, Roger Zelazny, and China Mieville continue to provide solid inspiration for narrative

craft and worldbuilding.

The design document for this game (see Appendix E) has been a living document for the last several years and will no doubt continue to shift and change as the game develops. Future goals may be categorized and bullet-listed to kingdom come, but the primary focus must be marrying the artistic direction and technical craft together to tell the story of the game. And, of course, to make something fun to play.

List of Figures

Table of Contents

Chapter 1

Introduction

1.1 Statement of Intent

Games create an active dialog with the player by using the mechanics of the game as constraints that form a separate reality for the player to explore. According to Dutch Anthropologist Jonathan Huizinga, this play space must be separate from the outside world, and part of the nature of the game's play space is that it is utterly absorbing, takes place in its own time and space, and proceeds according to its own rules. In *Homo Ludens: A Study of the Play Element in Culture*, Huizinga claims that play is less a mindset and something more akin to a sense; "[games']rationale and their mutual relationships must lie in a very deep layer of our mental being." After all, play exists biologically in a way that predates civilization. In the animal kingdom, a lion cub will play with its siblings by biting their ears and wrestling. It's important to note that their bites never cross the threshold from play to pain. Huizinga's work recognizes that this threshold ensures the game's nature does not escalate beyond "not serious," which is another criteria in defining what is or is not the nature of a game. The moment harm is done by a lion cub to another, play has ceased and the interaction has become something else, something not-playing. Play as a concept appears to be so universal that it is practiced by multiple species and is so ingrained in the nature of the living world that it may be categorized as a sensory experience like touch or smell rather than a pastime or instinct. [21] There have been many attempts at defining games, several of which are well covered in *Rules of Play: Game Design Fundamentals*. Typically these definitions are broken into lists of traits or inclusions that define what a game is or does.[47] While game design shares many

attributes typically associated with visual arts and graphic design, a large part of game design lies outside the visual realm. Because of this extension beyond visual design, it is necessary to explore what constitutes a game in order to create one. It's worth noting that like any other art form, a great game requires sincere dedication to design and craft.

Much like traditional design, the constraints or rules determine the methods and processes of the creator as well as the nature and quality of the game experience for the user. Unlike most other media, games invite the player to become participant. A game can be a multi-million dollar production created by hundreds of team members, or it can be a simple twilight backyard pastime invented by a couple of eight-year-olds drawing in the dirt with a stick. It doesn't matter who makes a game, it only matters who plays it. And to attract participants to your game it must be fun. Determining the attributes of a game that manifest fun can be difficult.

Defining fun is a little like defining a game itself: it's a multifaceted attempt to pin down a highly variable construct. The goal of Knight of Drones is to create a fun game that affords the player multiple modes of gameplay. By combining a top-down shooter with a side-scrolling platformer, the player has a variety of play styles and a greater number of methods with which to explore the world of the game. Altering the player's perspective within the game is a novel way to break repetition and introduce simple new mechanics without complicating the play experience.

Hybridity of gameplay may be utilized to break up the game loop, to progress the narrative, to surprise the player, or to strengthen the effect of the game on the player. My goal is to create a game where different gameplay modes speak to one another as the player solves puzzles and defeats enemies across different play styles. By building shifts in perspective and play, different narrative events can be weighted and scaled to different effect.

1.2 MDA Framework

Game design is an abstract process that requires the use and understanding of several standardized terms before discussion can take place. Games vary widely from tossing pebbles in a cup or kicking a ball to a hyper-real Western with a hunting system in a massive open video game world. Becuase of this range, game creation is perhaps best seen through the prism of Mechanics, Dynamics, and Aesthetics, or the MDA Framework. There are many ways to study game design. By approaching a game from this framework, a game can be broken down into components that are

easy to recognize and define using a standard set of terms to clarify elements of game design.[22]

Mechanics are the methods by which the player can interact with the game or other players. In poker, for instance, the mechanics are the cards, their values, the colors red and black, the four suits, the taking of turns, and the rules that determine which values or combinations of cards defeat others. Mechanics are the foundations of gameplay and the means by which the game designer creates the player experience through which the players enact their presence and decisions within the world of the game. Without Mechanics, the game would break or prove unplayable.

The next framework term is Dynamics. Dynamics emerge from player behavior through successful implementation of the mechanics on the part of the game designer. Gambling is not necessary to play poker. Gambling and all its nuances, such as bluffing that you have the third eight or slow-playing the best hand because you need to bleed your opponents who would otherwise fold, are examples of dynamic systems that emerge out of the human play of poker. It is not through the cards in this case, but through the collective use of the afforded mechanics that dynamics emerge. Gambling is not a mechanic. Though it has its own rules and mechanics, it is an emergent behavior and thus is considered a dynamic. Another example of a dynamic occurs in the board game *Clue*[38]. A player lying to throw an inquisitive family member off their trail is not a mechanic; it is behavior that the player chooses to engage in during play time. The rules of *Clue* create new behaviors in the players, who choose how creative or sly or witty to act. Dynamics reveal themselves at runtime, or during gameplay, as an effect of the player using the mechanics and/or interacting with other players.

The final term in the framework is Aesthetics. Aesthetics encompass the emotions evoked in the player by gameplay, which are usually influenced by the look and feel of the game. Aesthetics are often what the player first encounters and first envisions when asked to describe any given game. The aesthetics of *UNO*[41], for instance, are undeniably simple and instantly recognizable: primary and choice secondary colors, white text over black with a design as stark as brutalist architecture. The design choices are clear: stark design for a brutal game that's both easily readable and instantly playable. Aesthetics are the design choices that are not game design; they are the visual and audio cues that players use to inform their decisions and actions. Like in *UNO*, aesthetics should create or at least parallel the tone of the game, and the aesthetics should enhance the ability of the player to understand what's happening in the game. Aesthetics conjure the tone of the game in the mind of the player. They also keep the gameplay flowing by helping every player understand the current

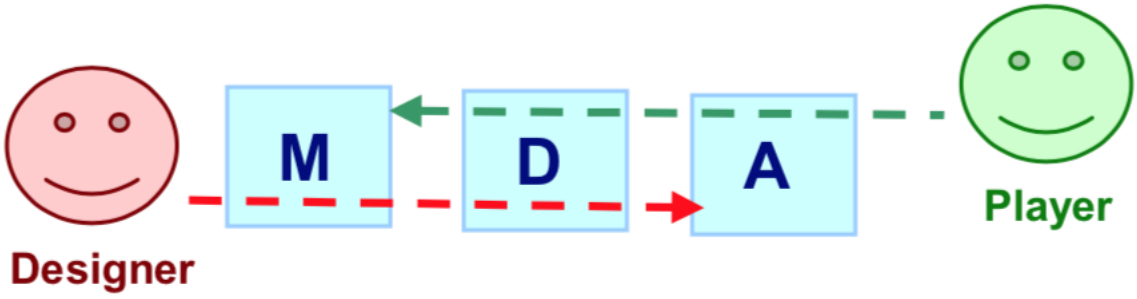


Figure 1.1:
MDA Framework from Designer and Player perspective

state of play and help them formulate their next decision.

Because players experience the game through the aesthetics, this aspect of game design understandably often receives the majority of the artists' consideration and approach. The first thing the player encounters is the splash page, the front of the box, or an opening cinematic. Aesthetics are forward for the players, then dynamics, and lastly, the mechanics are there to be picked apart by especially inquisitive players. Artists need to understand, though, that for the game designers this flows in reverse order. Mechanics come first in the game design process, followed by Dynamics (through playtesting), and lastly, Aesthetics. This is why the framework is ordered M-D-A.

Despite its placement in the framework's order, art and visual/audio design is still unquestionably important, and the order of operations for game design is not one of clear hierarchy. Each step of the framework needs attention and aesthetics often requires a large portion in the artists' skillset. Artists seek unity in their pieces. Artistic unity occurs when all the elements present in the work successfully combine to achieve the central goal of the work. No matter how disparate or fractured, even the most postmodern of postmodern conceptual pieces use elements of design and composition to achieve unity. In films, games, and other multidisciplinary arts, unity becomes increasingly more difficult to achieve as the scope and vision for the work expands. For the traditional artist, aesthetics is its own facet; however, for the game artist, aesthetics must achieve an ideal match with the game as a whole. The look and feel of the game should match the goals and tones of the mechanics and dynamics. As production design serves the story of a film, artistic direction and design choices serve the gameplay as well as the tone (or narrative) of the game. This may

sound simple, but small details and decisions become more complex when considered within the larger context of the game. For example, imagine choosing the font for the options menu for a first person shooter or selecting the color of HUD elements. Would one choose red for health and blue for mana? Perhaps green might represent health, yellow signifies stamina, and red is toxicity that will result in player death? If it's a bloody game or there's a red team in a multiplayer match, red would become confusing for toxic effects and the designer may decide toxicity should be purple as this appears less frequently in other areas of the game. Each visual cue impacts readability, which ripples out into the player's understanding of the game's mechanics. The player must have fast, accessible information about the game state so they may decide how best to make choices. Unity becomes more than design elements working together to create a meaningful or beautiful composition, it also necessitates clear communication of the mechanics to the player. The player must not be confused about what mechanics are active and available at any time during gameplay. Design must serve the game as well as the player.

1.3 Video Games and Psychology

Drawing inspiration from the rule "drama is conflict," games are often built with challenges or forces in opposition to the player. A small subset of games in general, video games have expanded into new interactive categories that warrant further study. Contrary to speculative media claims that violent video games (VVG) contribute to gun violence in the United States[18], shooters and action games appear to have net beneficial effects on the brain. Action gamers are shown to have better eyesight, increased ability to resolve different levels of gray, and increased speed at solving Stroop effect style challenges, where incongruous stimuli cause a delay in naming tasks.[6] Studies concerning the effects of VVG on behavior show that VVGs do not prime humans for violent behavior[52] and that family violence and innate aggression are better predictors of violent crime than exposure to violence in media[13]. In short, VVGs do not increase violent behavior and in fact seem to be having beneficial effects on the brains of players. The act of making a game that contains conflict or violence does not mean that designers condone real world violence. Provided that the game includes narrative elements to explain the reasoning behind something like shooter mechanics, a game can include combat without actively promoting any real-world use. Even better, games may be designed as positive influences for change and can be as much a powerful source of social commentary as films,

visual arts, and other forms of media. With these considerations in mind, the shooter mechanics in Knight of Drones serve a purpose beyond mindless destruction and add a layer of commentary for the player that does not cause Ludonarrative Dissonance[20], a term for when a game's message does not match its mechanics. It was a deliberate decision to make all the enemy NPCs in Knight of Drones robotic, and, later in the game, choices presented to the player concerning the death or killing of organic combatants are relegated to player choice and not forced through mandatory kill-to-progress situations. In this way, the goal of the mechanical systems of Knight of Drones is to present a narrative-based shooter that does not require killing to progress.

1.4 Vintage-to-Modern, Vintage within Modern

Vintage games present excellent gameplay models to inspire contemporary game designers in early development cycles. There is a rich history of gameplay mechanics with which to experiment, and many successful modern games are built on foundations established in eras when limitations of computation and memory constrained game designers to certain technical boundaries. As is often the case with artistic endeavors, boundaries can become a source of liberation and creative problem solving. Early gameplay mechanics and level design are no less immersive and captivating than modern games, even if their color choices and sound effects were limited by the hardware at the time. More than nostalgia, the purpose of selecting vintage game mechanics as inspiration is rooted in a long history of successful, enjoyable games, some of which have aged beautifully and are still worth playing today.

Knight of Drones combines familiar mechanics from older games to create a style of gameplay that feels new. Merging two primary game modes, a top-down shooter and a side-scrolling platformer, creates a memorable experience that feels simultaneously familiar and unique.

Chapter 2

Related Work

2.1 Vintage Roots

The primary inspiration for the sidescroller aspect of Knight of Drones is Rare Ltd.'s *Wizards and Warriors*[28] for the Nintendo Entertainment System, which is known outside North America as the Famicom. Released in 1987, *Wizards and Warriors* reviewed well, earning nominations for Best Sound and Music and Best Character (Kuros) in The Nintendo Power Awards '88[34]. In this game, many young, novice gamers found a captivating adventure unique from an ocean of Mario-clone sidescrollers because of its odd approach to level design and jump-heavy, exploration focused gameplay.

The mechanics of *Wizards and Warriors*' player controller are now-standard d-pad Up/ Down/ Left/ Right with jump/attack actions along with duck and duck attack actions. The player adds to this repertoire of player controller mechanics as the game progresses by adding items to their player inventory via chests strewn throughout the game's maps. Chests are often colored and locked with a corresponding colored key. Items like the Feather allow the player to press Up on the d-pad and float a bit higher. This new height can be combined with a jump to reach greater heights. Other noteworthy features include staves that shoot projectiles and the Boots of Force, which allow the player the kick open chests whether or not they have the proper key. Pickups that appear at certain areas of the map have various effects, such as a clock that stops time momentarily or colored potions that increase the player's jump height, speed, or constitution. The player has a health bar that, when depleted, leads to death, and the when the player health bar reaches a near-death low,

the music changes to a particularly attention-grabbing repeating melody to warn the player that they are about to die. The end boss has its own Enemy health bar at the bottom of the Heads Up Display (HUD) that lies dormant and empty throughout most of the levels and fills up during boss fights, which is reminiscent of Castlevania games.[26]

Figure 2.1: Vertical Level

A noteworthy design choice that proved to be especially inspiring is the verticality of the levels. While most platformers of the day built levels that moved the player left and right to progress, *Wizards and Warriors* tended to build extremely tall levels that would lead to perilous climbs and long falls. Mistakes by the player leading to a fall could set them back minutes at a time as they watched their knight fall depths that would need re-climbing. Add to this the player character's falling animation, which progresses from a simple fall to a back-down plummet with limbs in the air along with the slow, soupy air control on the way down and you have some particularly hilarious dynamics as the player attempts to salvage their footing and aim for any platform below. Forced falls complicate the game further by creating scenarios in which players bounce or slide, panchinko-style, to different platforms on the way down the map in an attempt to optimize item collection or enemy avoidance and minimize player damage on impact when landing. These mechanics are placed early in the game levels so as to prime the player for later levels, such as the notorious castle exterior climb with block platforms that animate into and out of the castle walls, forcing the player to time their jumps more carefully. All of these features create memorable layers of difficulty to add to precise jump puzzles.

Wizards and Warriors also features an overworld map to give the player a sense of where they are in the overall game's level structure, a common feature in early adventure titles where levels are beaten and the next level loaded with no visible connection between them. The map serves to stitch the levels of the game together in the mind of the player at a time when levels needed to be kept separate for technical limitations or production needs.

This overworld map mechanic is also present in Capcom's *Ghouls'n'Ghosts*[9], a notoriously unforgiving arcade quarter muncher and a

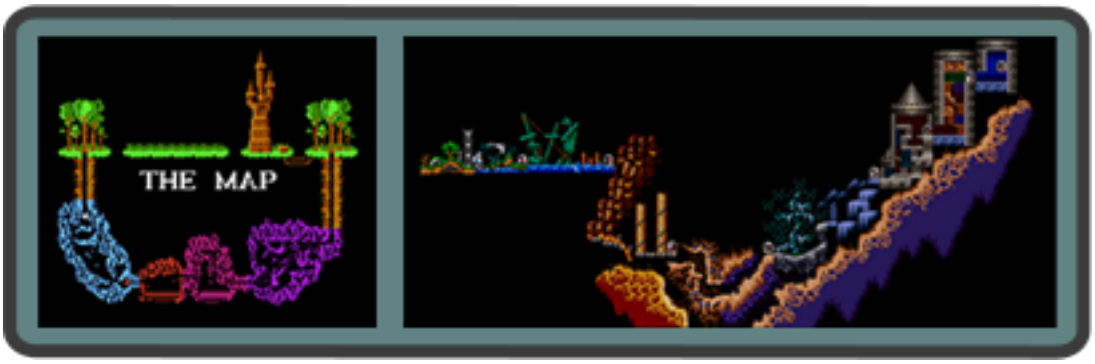


Figure 2.2:
Platformer Overworld Maps: Wizards and Warriors (left) and Super Ghouls'n'Ghosts (right)

more polished sequel to *Ghost'n'Goblins*. This overworld map appears after successful bossfights lead to a new area, and it also draws onscreen after each death to show the player's progress. Showing player progression in a game like this, with long, sprawling sideways levels and incredibly varied environmental hazards, helps keep the player excited about progress even as they are relentlessly murdered by hordes of enemies spawning in dastardly positions. *Ghouls'n'Ghosts*, built for arcade money collection, relies on a two-hit health system. After the first hit, the player spritesheet changes from an armored knight to the player character, Arthur, running around half-naked in his boxers. One more hit and Arthur turns into a pile of bones. Powerups to Green and Gold armor increase the effectiveness of the player's equipped weapon and add a charge up attack. These armor powerups do not increase player health; one hit and it's back to Arthur running around in his boxers.

Overworld maps serve to link the levels together and give the player a sense of where they are in the greater world of the game. Knight of Drones takes inspiration from 2D side-scroller mechanics and seeks to replace the overworld map animation with another game mode, allowing the player to play across the world of the game in the form of a top-down shooter. The player can opt to eject from the shooter vehicle to traverse the map and enter a different level, which loads as a side-scroller. Switches and encounters in the side scroller levels affect the greater overworld map, opening new areas to explore and eventually expanding the gameplay loop to include new items and abilities. The level design for the 2D platformer sections of Knight of Drones borrows heavily from the verticality and precision jump puzzles of *Wizards and Warriors*. The player controller is also built to mimic the feel of controlling Kuros, which has a distinctly slower, more deliberate feel to precision jump puzzles

than many left-to-right side-scrollers that tend to favor speed and memorization over exploration and discovery. Air combat in *Wizards and Warriors* is an interesting choice in mechanics, as Kuros cannot swing his sword in the air and cannot change direction. This makes the direction the player chooses to face before jumping quite important as the only way to kill enemies in the air is by facing them with the sword while colliding. As this is a defining characteristic of the *Wizards and Warriors* play style, initial mechanics for the knight's jump were planned to implement this system; however, upon further playtesting of the *Wizards and Warriors* jump mechanic in Knight of Drones, it was determined that changing which direction the knight is facing while airborne should be included, and future implementation will add an air attack animation to make air combat less passive and more player driven. The shooter mechanics of Knight of Drones have been modeled after early levels of arcade style shooters, with single-hit enemies making up the majority of the early game and mini-bosses and boss encounters following later. The strategy component of the shooter portion is currently limited to a type of castle defense, which needs more implementation of AI to differentiate between main target (player) and secondary target (castle).

Hybridity of game modes is nothing new, with several vintage gaming examples from which to draw inspiration. An early NES title, *Kid Icarus*[32], begins as a vertical platformer climb and grows into a maze-like, Zelda-style[31] dungeon crawler before its brief finale as an Irem- (creators of *R-Type*[23]) style sidecrolling shooter. The play styles change as the player progresses through the narrative: as the player grows in skill, the game shifts and presents the player with a new play method that suits the next portion of the player character's journey. Aside from a game manual provided in the cartridge box or a brief flit of text, there is no warning that the game is about to change. There are no tutorials and no special instructions. The player has the same controller input options as before, but now that the perspective or controls have changed, the player knows innately that the mechanics are new. With next to no practice the player is able to adapt to the new control scheme and continue on with no break in flow.

Another strong example of hybridity of game mode is *Herzog Zwei*, a combat/RTS mashup for the Sega Genesis.[46] Nearly impossible to play without reading the manual, it featured an advanced-for-its-time mechanics set that could be played solo or as local split-screen player-versus-player (PVP). Each player flies around an RTS-style map in a top-down shooter mode and is able to reach high speeds with the caveat of required refueling along the way to any given destination. The player has a home base where they can build a number of units, such as tanks and infantry, which the

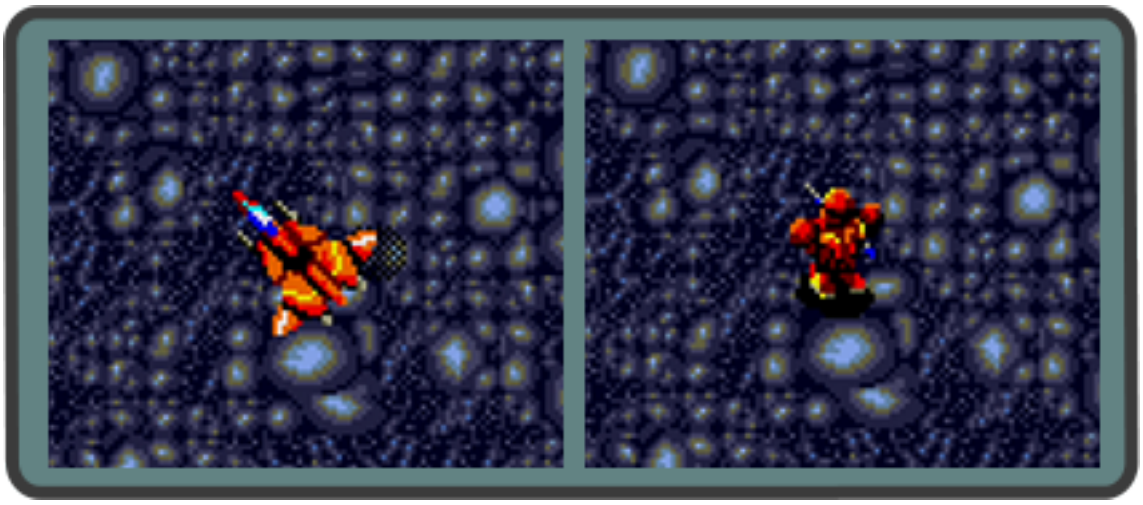


Figure 2.3:
Herzog Zwei Player Controller: Flying Mode (left) and Walking Mode (right)

player can then pick up with their ship and air drop to locations around the map. Each player's ship can also transform into a bipedal mech that traverses the terrain and fires weapons to damage any enemy units. When in flying mode, however, the player may only engage air targets. These features make it an interesting mix of real-time strategy and top-down shooting that continues to resonate with contemporary gamers enough to warrant its recent inclusion in the Sega Ages collection.[42]

A final inspiration and an excellent case study in game mode hybridity comes from Atari's 1980 arcade cabinet *Battlezone*[4] and its 1998 remake by Activision.[1] The original *Battlezone* is often remembered as one of the earlier examples of 3D graphics in the arcade. Its green vector tank combat is easily recognizable as a somewhat iconic image of the 80's arcade era. The original *Battlezone* is a fairly simple hovertank shooter that sets the user in a first-person cockpit and challenges players to wipe out battalions of enemy hovertanks. The gameplay is complex enough to be difficult, but it is not meant for extended play as the enemy variety wears thin quickly and the map can do little more than offer a flat horizon. However, the challenge of maneuvering to overtake enemy tanks as well as the visual feedback of low-velocity shells traveling onscreen to connect with moving targets in 3D space proves interesting enough to

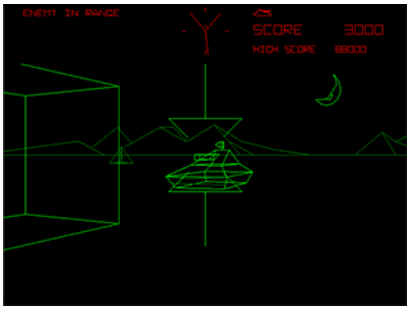


Figure 2.4: Battlezone (1980)

warrant play time and several generations worth of quarters. If anything held *Battlezone* back, it was repetition and lack of variety, and this is where the remake is worth study.

Now with texture-mapped polygons and early game physics models, the developers of the *Battlezone* remake, 18 years after the original release, created a convincing hovertank combat game with a compelling cold-war era alternate timeline story. In this world, the United States and Russia fight a top-secret war on the moon and, later in the game, on other planets in the solar system. This alone may have made a game worth some attention, but the developers decided to take the remake a step further and integrated a completely flexible hybrid Real Time Strategy (RTS) – First Person Shooter (FPS) gameplay combination.

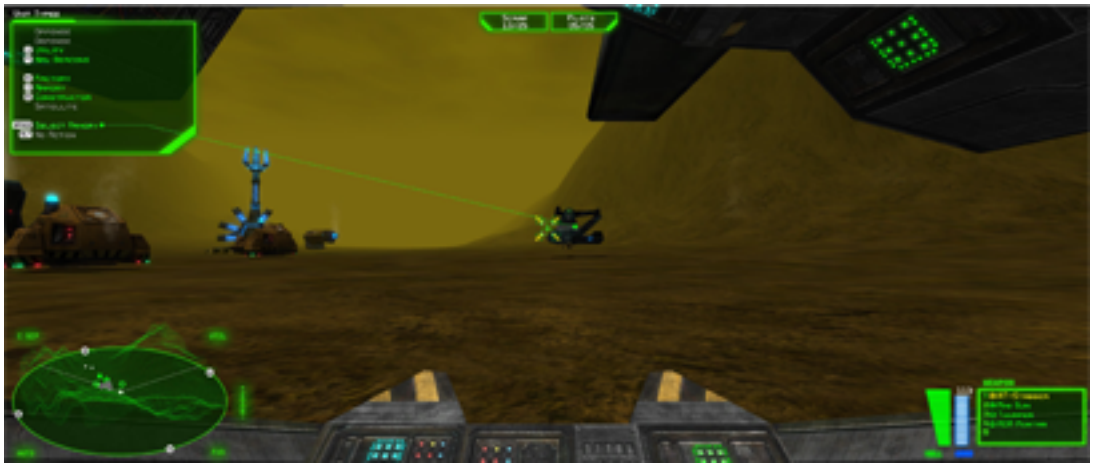


Figure 2.5:
Hybrid Gameplay Introduced to a Vintage Classic in Battlezone (1998)

Much like a standard RTS game, players collect resources, build bases, and fabricate AI-controlled battlefield units that they order around the map. However, in combining RTS and FPS gameplay, the player now does this in a realtime, first-person mode using a series of waypoint tags, combat targets, and reticle-based or map-based unit or building selection. Additionally, players are now able to exit their hovertank and walk around the terrain with the option to commandeer and pilot any other friendly vehicle in the game. If the battle seems to be going poorly or there is an order just a little too complex to trust to other AI-controlled squad members, the player can take the initiative and jump into a self-built artillery unit and clear a path for other units. Of course the player could also stock up on weapon types or ammunition and speed into imminent danger, leaning on their FPS combat skills to clear the area and secure victory. Alternatively, the player

could choose to send their friendly AI forces into an enemy-held area and observe and direct from afar, a heavy RTS-based approach. A game from 1980 that could have just used a face-lift instead recieved an innovative new way of playing an RTS game by combining it with an FPS, which allows for massive amounts of variety and player freedom. The repetitive nature of the original arcade cabinet has been replaced by a refreshing, memorable experience that was unlike anything players had seen to date, and not because any individual game mechanic had never been done, but because well-tested mechanics from multiple styles of play were being combined in interesting new ways. Two familiar gameplay modes had been seamlessly integrated together to create something new and memorable.

In Knight of Drones, the choice to combine a shooter with a platformer stems from a simple desire to create an interesting, memorable experience for new and old players alike. Because the inspiration for the platformer aspects of Knight of Drones is rooted in a slower, vintage platformer with a focus on careful jumps and wide exploration, it seems ideal to have these platformer elements placed in a top-down overworld map with a faster-paced game loop emphasizing speedy combat and fast dodge mechanics. This also provides an opportunity for Knight of Drones to grow interacting mechanics where the side-scroller portions directly relate to the overworld map. As an example, the end of a 2D platformer level could feature a switch that opens gates in the overworld, allowing the shooter portion of the game to progress.

Game mode hybridity is alive and well in the modern AAA landscape. Games with massive, varied, open-world systems often involve games-within-games: CD Projekt Red's *Witcher 3:Wild Hunt*[40] notoriously included the card game Gwent into the world's mechanics, much like Rockstar's *Red Dead Redemption II*[45] included Five-Finger Filet and Poker. This kind of hybridity intends to immerse the player further into the game world. Though, arguably, it can be a distraction or even a gimmick, these games-within-games serve to perk the player up and break the learned pattern of open world quests so as to stave off monotony. There's no reason not to put off that main quest to play a little poker in Valentine, for instance, as the game clock won't penalize you and it's easier on the head to exit a saloon at the crack of dawn after a fake full night of poker than a real one. The persistence of gameplay hybridity throughout the rich history of video games predates and often overcomes the need to control the pacing of a massive open world.

2.2 Modern Gameplay

However gameplay mechanics evolve over time, some trends have led to regression. Sometimes with increasing complexity, as evidenced by the increase in console buttons and pressure sensitivity, comes an increase in what the developers expect of the players. This can lead to some mechanics players find troublesome, for instance the need to hold a shoulder button, navigate through several touch-wheel menus to find an item, and release the shoulder button to equip. *Metal Gear Solid* games come to mind, as well as several Rockstar titles such as the *Grand Theft Auto* series or *Red Dead Redemption 2*. Sure, this mechanic works well enough, but it's often combined with a time-stoppage in game. This means that each time the player equips something new, the game world slows down. This can disrupt the assonance, or flow, of the game. *Red Dead Redemption 2* is an excellent example of how increased production values and state-of-the-art open world systems do not necessarily lead to increased player agency. *Grand Theft Auto V*[33] and *Read Dead Redemption 2* are notorious hand-holders, down to telling the player which button to hold when in a narrative sequence in order to perform an action that moves the game story forward. Hold left stick up to grab the wagon wheel. Rotate the L stick to roll the wheel back to the wagon. Tap A to bash the wheel back onto the axle. While this is going on, the characters are interacting in a beautiful landscape and each has their own animation sets that match perfectly the props and setting around them. It's all terribly impressive. But the player is being told exactly what to do at every moment. This is because the aesthetics and the mechanics are at odds. Rockstar has built so many systems and mechanics into their game that no one control scheme will suit every moment in the game. So when the narrative takes hold, the open world mechanics get placed on hold and new mechanics must be taught to the player as they happen. It's as if Rockstar wants a different hybridity of game mode: one where the player can play in their sociopathic sandbox of robbery and murder, and another where the player shuts up and does what they're told.

This can also go the other way, where instead of demanding more or less of the player, now the mobile or casual game may only exist as a pathway into the player's wallet. Many problematic practices involving whalehunting or dopamine targeting releases and pay-to-win models have opened up entirely new predatory practices in game development.[24] In this case, mechanics have shifted away from play models and into pay models. The dynamics that have grown from the mechanics are financial transactions that allow the player to continue their gameplay.

Bringing it back to practicalities: Another problematic development in modern game mechanics to be questioned (or perhaps even avoided) is the quest marker minimap. Similar to the Rockstar's need to hand-hold the masses through their wagon-wheel replacement cutscenes, with ever-increasing world sizes comes the need to help the player navigate them. What started as a clever way to lead the player to their objective has now become an all-too-familiar gameplay trope: follow the highlighted path on the minimap to get to your next objective. This isn't a good or bad thing, but it may be an overused one, and it is changing the way games are played to some extent. A comparison of Bethesda's *The Elder Scrolls III : Morrowind*[43] with their fifth entry in the series, *Skyrim*[44], reveals much about this modern gameplay trend. *Morrowind* featured no onscreen minimap or quest markers, instead relying on a journal system where key pieces of data were marked in text pages with hypertext-style links so that the player may jump between topics. Some players found this difficult when following directions to locations because a missed landmark could lead to stalled quests or getting completely lost. The trade-off becomes apparent with *Skyrim's* marker system, which, while convenient, also provides the player with almost too much guidance. The player now spends so much time looking at markers and minimaps that the dynamics of exploration and discovery that *Morrowind* was so heavily lauded for have been significantly reduced during runtime. *Skyrim* designers attempted to build the guidance system into the lore with an Illusion school spell, Clairvoyance, that will spend mana to illuminate a streak of bright mist along the player's path to their active objective. It's an admirable attempt to build a guidance mechanic that fits the dynamics and aesthetics of the game world, but the spell often sits neglected as nearby quest markers appear onscreen at all times, often hovering above doors or the heads of non-player characters (NPCs). The only way to turn quest markers off without modifiying the game is to turn off the entire Heads Up Display (HUD), which removes the ability for the player to see their health, stamina, and mana. Additionally, there is no guarantee that the game content provided to the player will offer sufficient directions or descriptions to get to their destinations without active quest markers. This is an indication that the nature of the included content of the game has changed due to the inclusion of the quest marker system. This suggests fewer work hours spent creating content for quests as the player no longer needs to read about where to go next, but this may also be interpreted as less content being provided to the player as well as the content being developed as biased towards players who use the marker system that is all but baked into the game's fabric at this point.

It's a strong aim of the design choices of each mechanic in Knight of Drones to avoid these

modern gameplay regressions. The player should feel rewarded for exploration and discovery. The best eventual course for Knight of Drones might be to include a minimap, if only to facilitate strategy or keep the player aware of their location relative to resources and points of interest. But this should exclude the mechanic of directing the player to where they need to go next. Exploration of the game map will reward players who pay attention and remember when and where their path is blocked so that as they gain new abilities they return to the areas to progress. This system is well proven in metroidvania style games, and a good example of an independent developer trusting the memory and fortitude of their players can be found in a Steam library game called *Witch Hunt*[51]. This is a smaller game in scope, but remains an excellent example of what a solo developer can achieve with simple systems, a couple of maps, and curious players. *Witch Hunt* provides the player with single statement directives and a bare-bones tutorial before throwing the player into a brutal, terrifying forest full of deadly zombies and treacherous paths. The map is a single screen that updates only with sparse, select landmarks, and the level map is large enough to make remembering paths between areas a stressful survival horror experience in itself.

In this way, Knight of Drones avoids the dilution of vintage-style dynamics of exploration and discovery and pays homage to predecessors without patronizing or pandering to its player base. Questions of accessibility remain, however. Should the player be allowed to rebind their controls? This should be added to the to-do list, as each player should be able to change their button mappings to suit their play style or accessibility needs. Should the game be playable with a mouse and keyboard? An important question if the primary release platform target is for Personal Computers. This is a more difficult question to answer, as a top-down shooter by nature begs for a controller with analog joysticks. More research and playtesting will be done to determine if a keyboard and mouse setup is feasible for this game, and if at all possible, it should be implemented. However, to avoid the pitfalls of multi-platform feature creep, if the gameplay is designed for controllers, mechanics and difficulty should not be compromised to adjust for other input devices.

2.3 Vintage-in-Modern

It's worth noting that vintage gameplay is alive and well and several successful homages to older genres can be found in contemporary game libraries. *Shovel Knight* by Yacht Club games[15] is all but a sopping love letter to vintage Capcom and Nintendo 8-bit era games. As of 2018 it sold over

2 million copies across ten platforms.[16] It uses familiar mechanics from the era to great success, and, particularly in free Downloadable Content packs added after launch, proved that innovation is still possible even in bygone-era player controllers, level designs, and boss encounters. See Appendix A for a side-by-side comparison of select vintage mechanics in Shovel Knight with their predecessors.

Chapter 3

Design: Content, Tone, and Message

3.1 Drones and Technology

Knight of Drones takes place in a distant future after all humans have destroyed each other and all that remains are self-replicating factions of military robots and drones. It could be compared to the SkyNet future of James Camerons' *Terminator*[12] series but without the cautionary take on Artificial Intelligence, human faction fighting to survive, or the time travel. Instead, it aims to be a tongue-in-cheek take on renewed tribalism by having the military robots continuing to fight each other because it's what their long-dead creators programmed them to do. Much like the AI example of the paperclip maximizer[30], which states that if you build a machine to self-replicate and create only paperclips, eventually the entire mass of the universe will be paperclips or machines designed to make them, Knight of Drones is based on the idea that if machines of war are designed to fight wars, all they will see is battlefields. Add to this the speculative science fiction of an army of machines that was designed to self-replicate and you have the core idea of the game.

The mashup of medieval feudalism with this advanced technology provides a strong foundation of culture to draw from, and it also serves as a humorous anachronism: take humanity's bloody past of tribalism and feudalism and blanket it with technological adornment that has ironically outlived the people who created it. And yet in doing so gets stuck in a programmatic cycle of

constant conflict, the legacy of humans lives on! The idea creates ample fodder for fun combat as well as thought-provoking societal critique for any gamer who chooses to dig into the lore scattered throughout the game.

Figure 3.1:
Early Concept Story Frame

While this game is not intended to be a cautionary tale about the growth and use of technology, it is worth noting that there are several areas of development that have become alarming and problematic. It would be irresponsible to create content without researching the current state of ethics on the use of drones and robotics on the current fields of battle, and in some cases in areas of civilian life.

Setting aside several known areas of contemporary technological overreach, such as the rampant rise in facial recognition tracking and social media economics of contemporary China now bleeding over into U.S. cities[5], which brings a whole range of ethical problems, there are uses of drones in America that warrant some investigation.

Well known are the extrajudicial killings and assassinations perpetrated by the United States government[35]. These strikes often rely on data collection and vast systems of communication, and the news headlines often involve the victim and the perpetrator. But interestingly, studies have shown that drone operators suffer from PTSD symptoms[7], even so far removed from the real-

world effects that their experience could be described as a simulation often compared (somewhat dubiously) to a video game, such as in the 1992 film *Toys*, to use a cultural example.[27]

Another noteworthy occurrence of drone operation, this time in an American city, involved the apprehension of the Dallas gunman who opened fire on police officers from the top of a parking garage. Dallas PD pinned the gunman in a parking garage basement and notified the perpetrator that he could put down his weapon and surrender or be killed. The gunman declined, and Dallas PD sent in a bomb robot, which detonated and killed the gunman.[14] Understandably, people began asking questions. What is a bomb robot? Where did Dallas PD get it? How many more were there? Who was trained to use them?

Recently, the NYPD unveiled a new robotic dog[3], complete with blue paint and tailing officer with a huge remote control. Videos of the robot appeared online as the unveiling went viral. Questions concerning the ethics unmanned reconnaissance have been replaced with even more practical questions: should the police force have access to this expensive technology in a district of New York City that can barely afford to staff and supply its schools? Shiny technology and life-saving equipment may appear promising in a boardroom full of city officials. Perhaps if our students were all shiny expensive robots, they would be more thoroughly funded and all the classrooms would be filled with qualified, well-trained operators of shiny expensive robot children. Outrage over these questions led to the NYPD cancelling their lease and returning the digidog.[50]

In a tragic incident in South Carolina, a former NFL player murdered a family and then took his own life. A robot was used for reconnaissance of the area so that in the event of an active shooter, officers would be able to gather information about the immediate area without placing their lives in danger.[39] These events appear to be growing in number, and it seems possible that soon the military and police force will be utilizing unmanned drones as a first response. In many ways, this is a good thing. If robotics offers solutions that increase speed and safety, their use seems a natural choice. Imagine a squadron of drones programmed to fight fires or lift loved ones or pets to safety. Doing so without endangering the lives of staff could be a technological leap forward. But it's important to ask not just how these things can happen, but should they, and why?

The legislation regulating unmanned drone use varies by state in the U.S., but there are federal guidelines in place for Unmanned Aerial Vehicles (UAVs).[2] Globally, it is illegal to create killer robots.[8] Thirty countries have expressed desire to draft a treaty banning their creation.[48] These laws have been created to prevent war crimes that may occur in a war where one side is

human and the other completely robotic. Wealthy countries would be able to fight without putting their lives on the line, limited only by their resources and ability to deploy. The other side could very well face massacre with opposition loss of life remaining at zero. But has the discussion now shifted into old rules of engagement, and is human life now measured in a different way? If one side has the ability to fight a war without risking the lives of its citizens and military, does it not have the moral obligation to do so in order to minimize losses?

These questions seem new, but in many ways they seem similar to any questions about conflict since humans have existed. For this reason a game where robots fly banners and defend castles is not only hilarious, it is scarily relevant and definitely a game worth developing.

3.2 Environmental Message

Knight of Drones is inspired by vintage gameplay, but deviates from known vintage game tropes. Early platformers present the player with common themed levels: the ice level, the lava level, the desert level.

In place of these vintage motivations for level aesthetics, Knight of Drones will feature levels that draw from humanity's abuse and neglect of its environment. Players will navigate the Plastic Refuse Island level, or the Abandoned Oil Tanker level. Some aspects of level design should reflect nature's ability to reclaim land, so the game world will not be devoid of foliage or fauna. But several areas should read to the viewer as permanently ruined, and as the difficulty of the game increases the levels should parallel the increase in difficulty with an increase in the visible effects of a degraded environment.

Several real world locations will be drawn from for this dark inspiration. The Carrie Furnace[36], a derelict iron forge in operation from 1884-1982, provides views of rusted metal towers and tapering, connecting vent systems. This refinery could serve as inspiration for a robotic army's self-replicating facilities and is well suited for level assets that need to bridge medieval structures and speculative, ruined far-future ones.

The Nike Nuclear Missile Site S-13/14[49] is a Cold-War era missile launch complex near Seattle, WA that's been abandoned since 1974. The area is now a protected park, and there are tours of the maintained areas of the complex. Much of the surrounding space is littered with trash and refuse and overgrown with vines. It could be an interesting overgrown bunker level or an interior

warehouse that abuts an exterior landfill level. It could also provide architectural reference for an underground bunker level that would appear later in the game.

The man-made waste runoff lake of Baotou in Inner Mongolia[29] is a dark toxic slick that permeates the area. A toxic body surrounded by Rare Earth mines and refineries, the environment at this site provides a direct link between our technology addicted consumerism and its dark, destructive effects. It seems apt to design a level around drainage runoff that results from harvesting minerals that go into our smart phones and televisions, as well as the very computers we use to create artistic assets.

The common tropes of vintage game levels did not come to be due to aesthetics, of course. They exist because they afford the game developers motivated reasons to increase difficulty of mechanics. Ice levels commonly introduce slick platforms with lower friction, causing the player to slide along the ground unpredictably. Lava levels usually spew fireballs or threaten players with flaming pits of death. So as the game increases in difficulty, Knight of Drones will use environmental themes to similar effect. The black sludge of the technological production runoff could serve as this game's ice platforms, decreasing the ground friction of the platformer game controller. Platforms of plastic refuse could crumble below the player's weight, leaving them small windows of time to jump away before they fall to their doom. These common vintage level mechanics are familiar, but repackaged with a new purpose of aesthetics that serve the cautionary message of the game.

Chapter 4

Design: Lookdev and Asset Generation

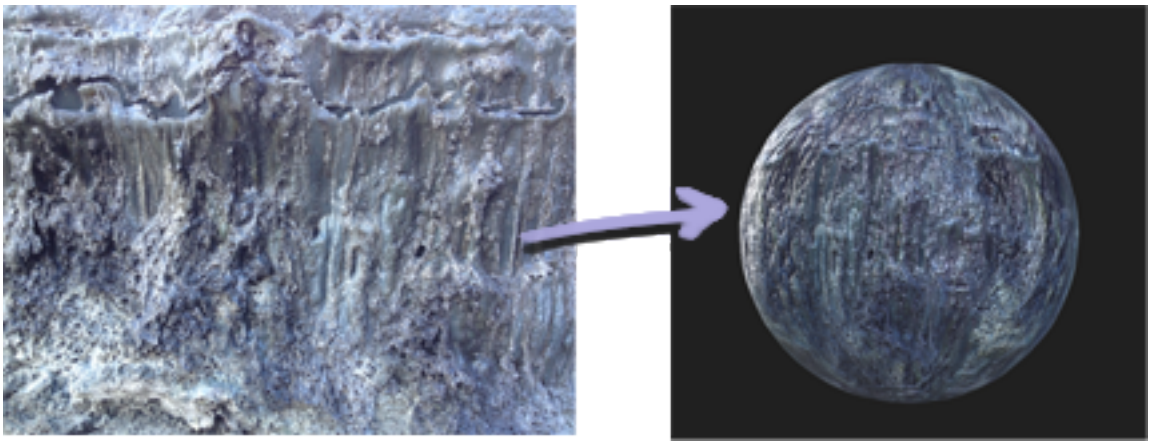


Figure 4.1:
Bitmap to Material Melted Plastic Example

The overarching design philosophy of Knight of Drones is to mash up medieval era arms and armor with speculative science fiction mechanical design. Level design elements will feature stacked stone and pounded iron elements, while character and NPC designs originate from contemporary hard surface design and weathered mechanical attributes. Sources for enemies that the players encounter may be remniscient of Boston Dynamics robot[10] gone awry, or of terrifying, out-of-control Vine Robots[19]. Shades of military style drones should come to mind as players encounter

them. Assets will be surfaced using Physically Based Rendering shaders so that materials will respond to in-game lighting in realistic ways. To enhance mechanical design, emissives are utilized to add glow and flash to create the look of an internal energy source. Painterly, stylized textures are to be avoided and whenever possible assets should appear to have weight and believable material properties. A majority of the game actor assets for Knight of Drones will have an aged hard-surface look, and the game features a ruined or desolate organic landscape, so the fidelity and lighting possibilities of a PBR workflow are better suited to this game than a painterly, stylized approach.

Additionally, level assets take on a melted plastic layer as the player progresses, increasing player unease and highlighting dangerous areas. The plastic materials were created using Allegorithmic's Bitmap to Material and Substance Alchemist. Photographs taken from a nearby plastic bin storage warehouse fire served as excellent material sources.

4.1 Knight

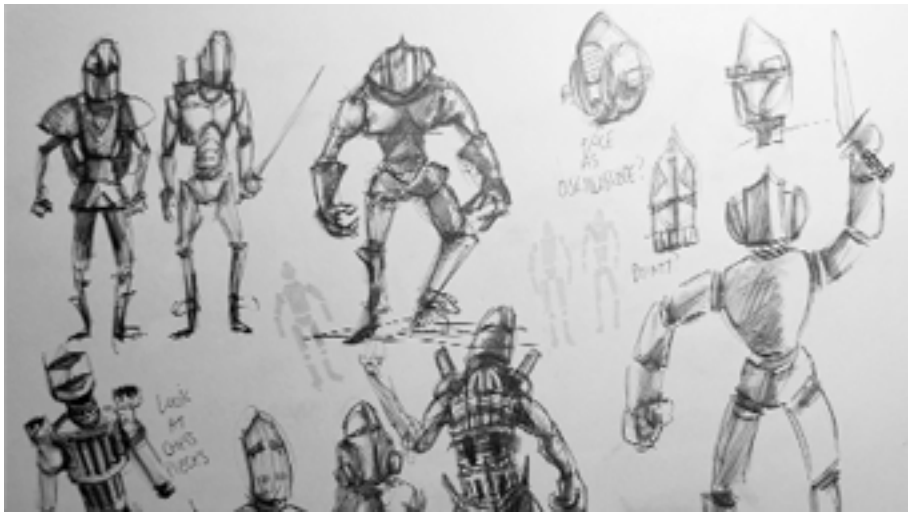


Figure 4.2:
Early Knight Sketches

After several rough sketches, the Knight began as a stand-in asset box-modeled in Maya, where it was then rigged and skinned with a HumanIK preset. The rig was animated with a simple walk/run/idle cycle and a quick attack animation. The model was then UV'ed in Maya and textured in Substance Designer before being shipped to Unreal Engine. This stand-in Knight asset served

as a practice run for shipping an asset to Unreal and building a basic animation state machine and player controller using the UE Character class. Once the trial run of the asset proved successful, the design of the main asset moved forward. The current, polished Knight asset is the main player controller skeletal mesh and represents a significant effort to create a working hero asset for the side scroller portions of the game. Initial sketches sought to strike a balance between medieval and hardsurface mech design, and further research and development led to a primary design document, which features a turnaround, pose ideas, and multiple stylistic references.

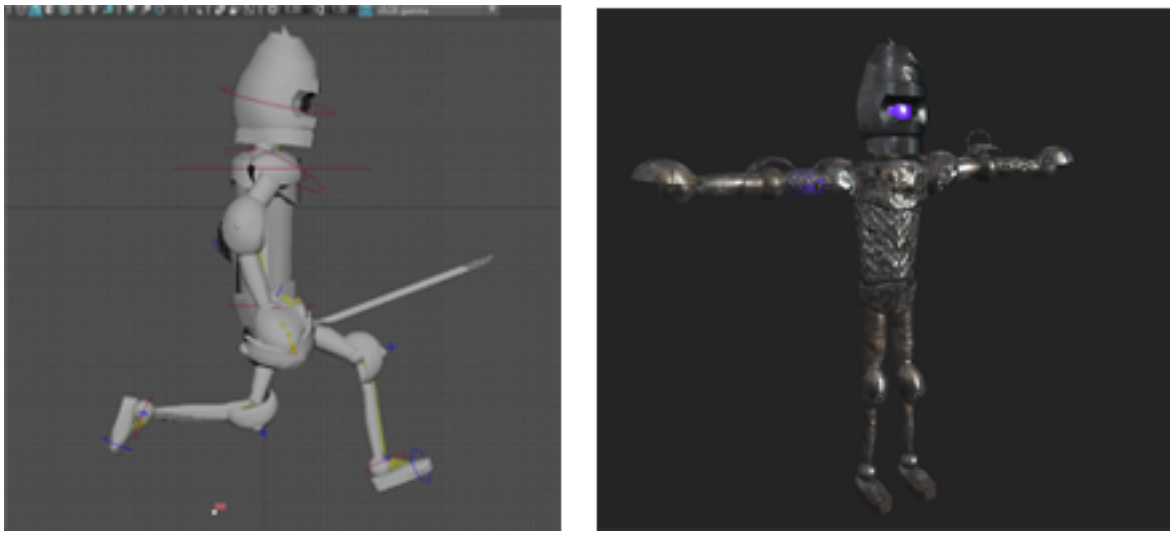


Figure 4.3:
Knight V01 Run Cycles and Texturing

The Knight is a primary hero asset that will serve as the mesh for the player controller during side-scroller gameplay, and it embodies the mashup of medieval armor and technology. It needs to look worn and embattled and feature a strong, recognizable silhouette. The Knight is a main character that should function to stand out from the game levels and have enough appeal to fit center stage on a poster or one-sheet for the game. It is a mech at its core, so it is designed as a hard surface mechanical robot with a layered appearance.

An interior endoskeleton featuring the robotic workings of the mech, including spring details, bolts, and smaller plated armor details give the impression of a functional inner structure, while the outer shell was designed to evoke the silhouette of a knight by expanding the design to rounder, more organic forms and increasing size and amount of armor coverage. The helmet proved a difficult

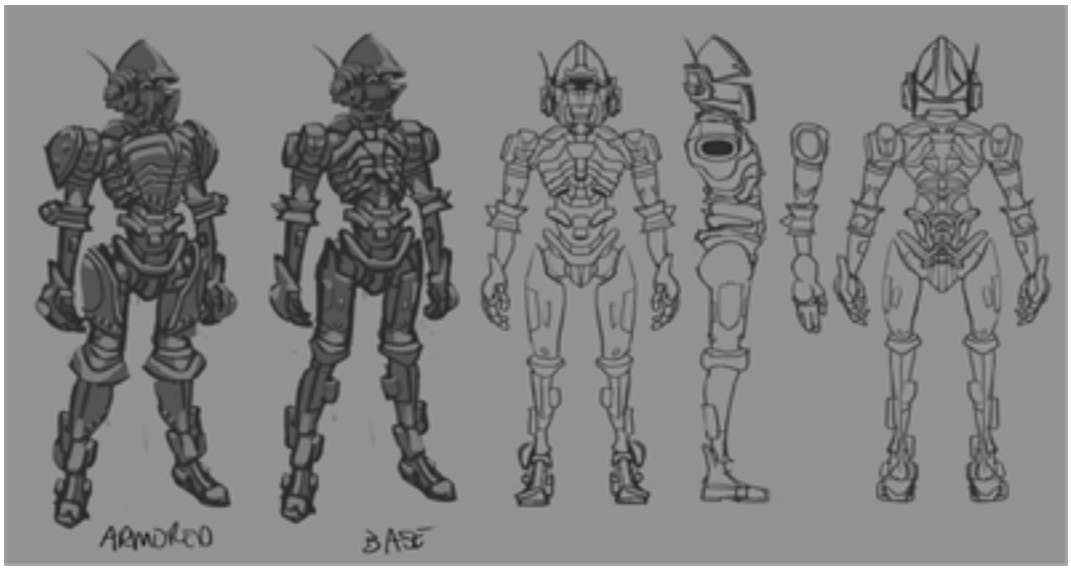


Figure 4.4:
Knight V02 Design and Turnaround

design to nail down and went through at least 5 iterations. Some appeared too mech-like and others didn't bring recognition of a knight's helmet. Finally a simplified design of a knight's traditional helmet was referenced to create a recognizable silhouette with clear facemask and jawline. An eye guard was designed with a pivot that would allow the guard to hang at the back of the helmet (at rest) or be rotated forward, covering the face. The interior of the head is a base sphere mesh that has been kept simple with the goal of adding an emissive texture to represent the face. In the future, it would be ideal to wrap a small animated texture over the interior head so that the player can gain additional feedback from the player character. An indication of eyes or expression through a simple motion graphics animation of a blink cycle or other expressions would increase the personality of the player character.

Thinking forward to the rigging and animation process during early design and throughout modeling proved necessary. Spherical joints were chosen to maximize joint rotation. At the shoulders, cylinder forms sprout from the sphere joints to distance the appendages appropriately to minimize polygonal overlap while animating. At the knees, the armor curves out away from the mesh to provide room for joint rotation. Hoop details at the forearm and rear of the knee broke the silhouette into a more interesting shape and also provided some details that could be animated with secondary action to give more weight and bounce to the character's walk and run cycles.

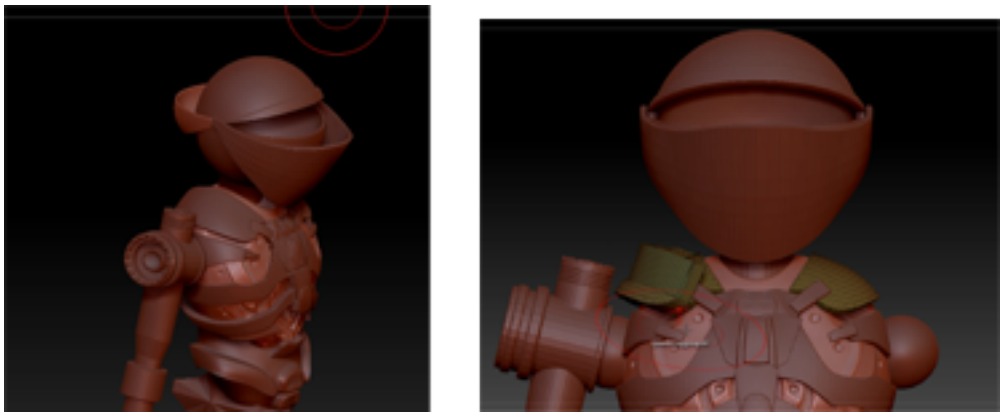


Figure 4.5:
Early Knight V02 Sculpt Based on Concepts

The sculpt for the Knight's base mesh was done in ZBrush using the design document turnaround as reference at first, then allowing for more improvisation and deviation as the sculpt progressed. The complexity of the hardsurface pieces grew throughout the process, which used the Zmodeler brush on the low polygon surfaces and then Creased Polygroups that were subdivided to maintain the crisp hard surface edges. The final Knight mesh ended up at just over 200 subtools, which were then merged into appropriate groups to prepare the mesh for retopology, texturing, and rigging.

Figure 4.6:
Knight V02 Final

A good amount of effort was spent experimenting with an auto-remesh/auto-UV method using Houdini's SideFX Labs tools.[37] The goal of these experiments was to create a section of the asset generation pipeline that would increase speed by negating the need to manually remesh and UV unwrap each asset. A GOZ transfer brought the high-res mesh into Houdini, then a Polyreduce node followed by a voxelization node created a closed-envelop mesh. Each group was separated with a split node and the voxelization applied with a for-each node. After the voxel mesh's resolution was adjusted to a proper level, an Exoside Quad Remesh node was added to clean up the mesh and convert the triangles to cleaner quads. Then UVs were generated with a SideFX Labs auto-UV node and UV visualizer. However, too much hard surface detail was being lost in the remesh process, particularly smaller live Boolean applications. It was determined that the auto-remesh and auto-UV processess were indeed useful, but should be limited to background assets and smaller elements that would appear at some distance from the camera. So this process will continue to be utilized for parallax background assets and overworld enemy types, neither of which will be subject to close scrutiny during gameplay. As described in later sections, the experiments were not successful for the Knight asset, but did lead to time saved in preparation of mesh assets elsewhere in the game.

Because the Knight is a hero asset and the primary player character mesh, a hands-on approach was worth the time. However, manual remesh via Topogun or Quad Draw utilities was determined to be too time consuming after the research time spent in Side FX Labs tools. Therefore, the mesh was decimated down from approximately 20 million to 220,000 polys using ZBrush's Decimation Master. This low res mesh was then brought into Rizom UV and unwrapped manually for each hard surface component. 220,000 polygons for a hero asset in engine is slightly heavy; it is recommended that a final remesh of the asset be performed in topogun to hit a target of 60-70,000 polygons, which is more appropriate for a game asset.

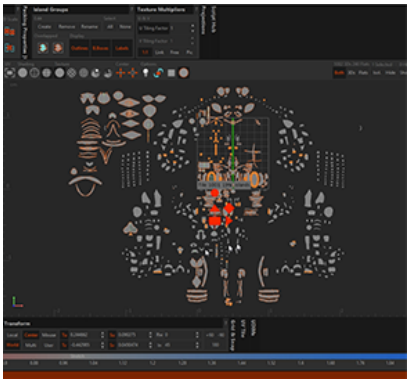


Figure 4.7: UV Unwrap Process

Using Maya, each hard surface piece was assigned a Lambert shader material group representing an ID map of different metals and types of hardware surfaces, which range from painted steel to rusty metal and rubber-coated wires. These 10 Lambert materials formed the basis across which to spread the UVs.

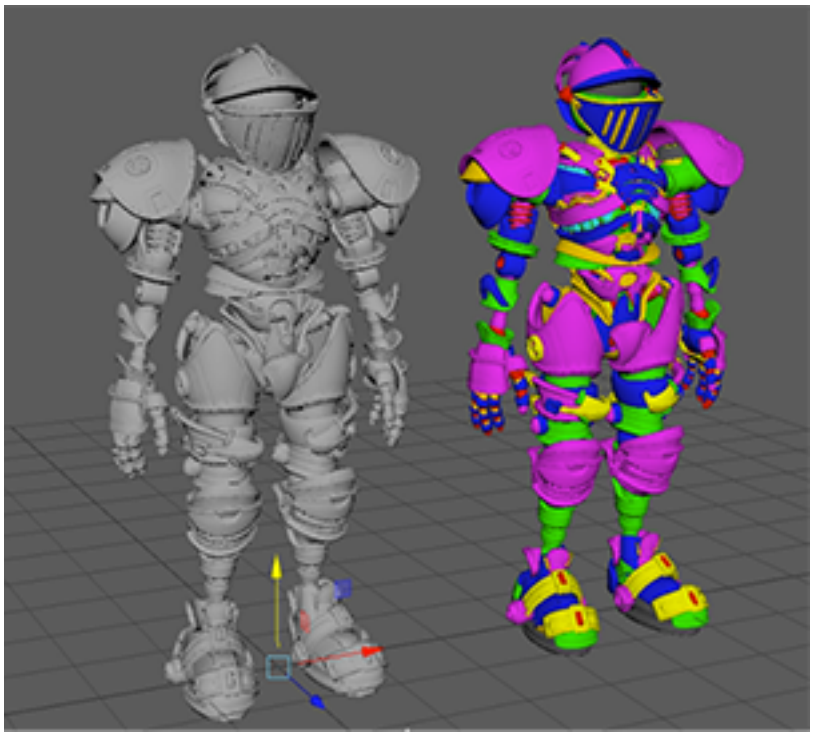


Figure 4.8:
Lambert Materials Applied to the Low Resolution Mesh

After some research into Unreal Engine's most recent implementation of UDIMs, the UV groups were spread across several UDIM tiles in Rizom UV and exported as a low-res texture-ready OBJ. This was brought into Substance Painter as the base mesh and UDIM tiles were activated in the new project. UDIM is short for U-Dimension and refers to a type of UV mapping that utilizes tile sets that extend outside of the standard UV 0-1 space. This increases texture fidelity or texel space because texture sets no longer need to be packed into to the 0-1 U and 0-1 V tile space. This workflow is typically reserved for film work as realtime engines haven't often previously supported UDIM tiling, but with Unreal Engine's virtual texture streaming activated in the project settings and a notorious tile packing bug from version 4.25 fixed in Unreal Engine version 4.26, UDIMs have become a viable option for Unreal users. This was a natural choice for the Knight textures due to the high number of separate pieces that make up the mesh. The Lambertian materials from Maya were selected for the ID map as the polypaint vertex color ID coming from ZBrush displayed a few too many gradient spills between some subtools, likely from user error keeping grads selected when applying the polypaint by polygroup.

Figure 4.9:
Knight Textured in Substance Designer

Initial texture maps were baked from a high resolution FBX rendered out of ZBrush down onto the decimated low res Substance Painter source file at a high dilation rate with maximum anti-aliasing (8X) in order to achieve the highest quality mesh map sources possible. Ambient Occlusion (AO) was baked with backside faces ignored and with self-occlusion active for the whole mesh so that the interior and exterior components of the hard surface details all contribute to a convincing depth and shadow overlap across mesh pieces. In the future, it would be ideal to organize mesh components with a naming convention that would facilitate self-occlusion on a more object-specific basis, but the current bake only shows AO artifacting on some close-up interior surfaces that won't be visible to the player. Standard and smart materials were used to create the look of the knight. Standard materials often create a nice base to work from, such as a rusty iron or matte plastic. Smart materials utilize the mesh map bakes (curvature maps, AO, thickness maps, etc.) to quickly generate areas where paint would wear off or where metal welds would swell. Smart materials save time and look great, although these can lead to a recognizable, almost generic preset look that needs to be adjusted. Altering base colors, scale and rotation, or random seeds can mix up the base smart materials, and even then additional layers of dust and grunge were added to age up

the surfaces and make them appear worn and used. Complementary colors were the target for the knight's look; a cool ash blue over chipped painted steel for the armor and a conductive-looking copper material was chosen for the interior and most hardware pieces. Complementary colors are pleasing to the eye and provide a nice base contrast for the player. The ashy blue chipped-steel paint look makes the armor appear sturdy and time-tested, while the copper should convey a more modern look and hopefully offset the medieval feel of the armor design with an uncommon tint for armor plating. Another set of structural details on the interior chest took a plastic that was initially painted with a Printed Circuit Board (PCB) green appearance, but the saturation level required for the PCB board proved too glaring for the rest of the mesh, and so the saturation was toned down and the material reconsidered as a type of interior plastic casing. Furthering the anachronism of futuristic medieval design choices, emissives were painted onto select areas of the mesh and over the face of the knight. The recently added ability to paint across UDIM tiles proved very convenient for this portion of the workflow. A very few materials were instanced across the whole mesh and then tweaked from there. A layer of diffuse dust applied with inverted curvature maps helped tone down some reflective surfaces that were too shiny and new. When needed, contrast filters and dirt generators were applied to material masks to break up their distribution and add age to selected materials. AO and curvature maps again proved useful as bitmap masks during the dirt and dust application, occasionally requiring brightness/contrast filters or inversion to properly orient dust and dirt buildup in crevasses. It's important to the aesthetics of the game that the knight character not appear shiny and new. Once complete, the materials were rendered out using the Unreal Engine Packed method that cooks out a base color map, a normal map, and an AO, Roughness, Metallic map packed into a single bitmap's separate Red, Green, and Blue channels. Emissives were included where applicable as their own bitmap. The materials were then imported alongside the mesh and hooked up in Unreal Engine's material editor. As the Knight asset will enter Unreal engine as an animated skeletal mesh, it must first be rigged and animated. The Knight was rigged in Maya using a straightforward FK joint hierarchy. Due to the hard-surface, mechanical nature of the mesh there wasn't need for skinning and deformation of geometry anywhere except the feet, which are large and flat enough to present a problem if remaining stiff while animating walks and runs. The feet were separated off the hierarchy and rigged with two bones, then skinned and weight painted to allow a bend at the toes. A hoop detail covering the top arch of the foot was modeled as a plan to hide this foot bend, and for the most part has proven successful. A future pass on the rig might add

a control to the hoop detail to allow it to rotate as the foot bends. Other secondary details were altered to have specifically placed pivots in order to be animated to create some secondary action. These include a lower, inner portion of the shoulder armor and a ring detail on the forearm exterior, which bounce and move depending on the primary action. Another important detail on the rig was the pivot placement and animation of the hip springs, which sit atop the hip armor at the waist and end in the lower abdomen. These needed to animate to avoid clipping into the legs and hips as those move; it also serves to show some indication of the inner workings of the mech and give some loose, bouncing mechanical details to show the knight is not a closed, tight system, but more a loose assembly of parts. The secondary animation details on the model are animated on the geometry instead of controls, this method proved speedy and kept the rig controls lighter and easier to select. Care was taken to freeze this geometry's attributes and place the pivots before animation began.

The animation cycles are straightforward in terms of action and looping, but the idea behind the motion was to give weight to the robot whenever possible. The walk is on 16's, the run cycle on 8's, and the idle cycle gives the impression of something alive and ready. The jump cycles were animated as an animator would tackle them: the jump start has an anticipation down and a slight upward momentum on the hips and the jump end has a collapse and compression of weight that returns to a standing ready pose that will lead back to the idle, walk, or run. The animation cycles were defined in Maya's time editor and the entire knight was selected and exported using Maya's game exporter feature. In-engine testing of the jumps provided feedback that the anticipation down and the up on the hips in the animated joints is unnecessary or needs reimplementation. The game engine provides the up, and so jumps need to be animated as if the feet are coming up off the ground instead of the hips dropping towards the knees. And while the anticipation down was known to be a chance worth taking, the player controller would need to delay the jump to play the animation properly, and a delay between player input and visual feedback should likely be avoided. Looking to the Wizards and Warriors version, there is a very slight anticipation down before the character jumps, perhaps more trial and error could lead to a solution that feels right

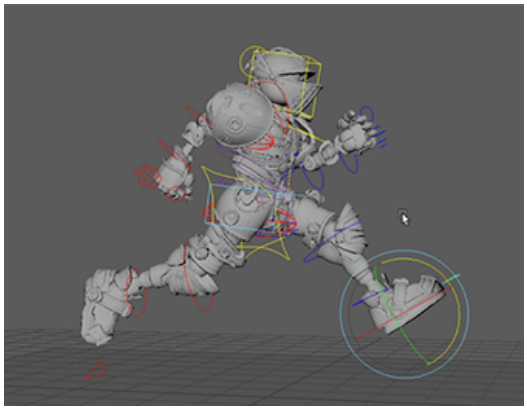


Figure 4.10: Knight FK Rig in Maya

for the player and animates the character in a more believable way. But for now, the animation frames at the beginning of the jump start animation have been cut off in engine so that the jump feels more responsive. Another pass on the jump animation could lead to better results with more study of references and some more careful keyframe planning. There is also an issue with the jump end, where the animation of the weight coming down can occur as the player is moving left or right at speed. This gives the impression of the Knight sliding across the ground. A solution might be to use speed to transition more quickly to the walk/run from the jump end, or to consider adding a jump end that accounts for speed and has the legs in motion already.

The animation state machine was implemented with a standard 1-D blend space for the Idle/Walk/Run and transition rules and bools for playing the jump up, inAir, and jump end states. The animation state machine is the link between player and animation. The player controller sends signals through the state machine and the game engine plays back animation according to the rules set by the state machine. For instance, there the bool inAir check prevents the walk or run cycles from playing while the player is in the air, and instead all actions while inAir is set to true must stem from the inAir idle (falling) animation.

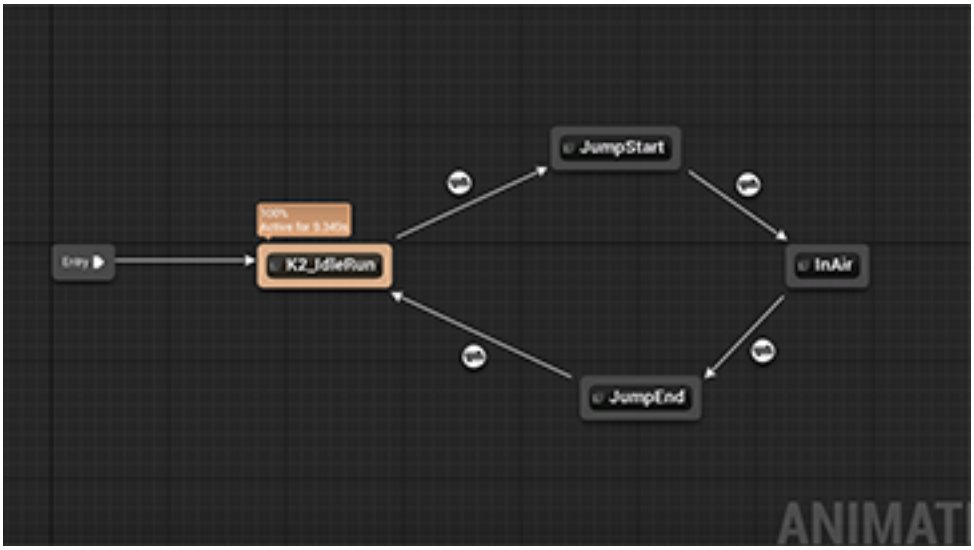


Figure 4.11:
Animation State Machine

4.2 Level Assets - Primary

The first side-scrolling level of Knight of Drones is a simple vertically oriented platformer stage. The level design began as a hand-drawn sketch which was then blocked out in engine to create colliders and floors that could be play tested with the player controller as early as possible. The intent of level one is to create a precise, jump-centric platformer level akin to an early Wizards and Warriors map. It was also designed to be an ideal play space with which to test and tweak the movement mechanics of the knight player controller.

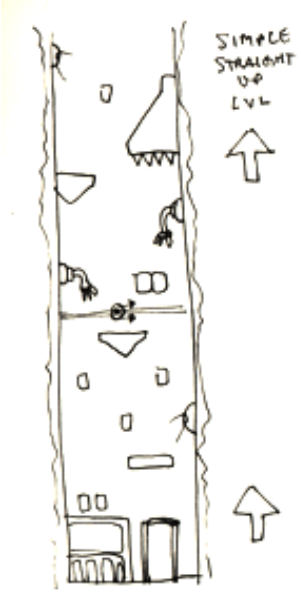


Figure 4.12: Vertical Level Sketch

The sidescroller Level 1 assets took advantage of the free Unreal Engine Marketplace plugin called Blockout Tools[25], which allow the user to create fast blockout collision placement for floors, walls, and platforms. These may then be selected in the engine and exported as an FBX. This full-scale level FBX was then brought into ZBrush and subdivided and sculpted into the primary art assets that appear in game. The design of the back wall elements gradually transition from stone and masonry to melted plastic waste, and the back wall features crumbled openings that serve as windows. This was a deliberate choice to increase the depth of the level and introduce parallax elements. Four major background elements were sculpted as bespoke assets for the level, and three sizes of platform "brick" were sculpted to be modular assets for placement throughout the level. The brick volumes selected from the blockout stage were roughly 1X1X4, 1X2X4, and 3X4X4, as these were the most common shapes in the level. Each brick was sculpted with a different front and back face to increase versatility. Each level piece was then UV'ed in Rizom UV and baked and textured in Substance Painter. The major background wall assets were textured to increase in plastic/oil darkness as the level increased in height so that the player would notice a progression from normal rock and stone to more melted plastic, ominous toxic surfaces the further up the level they progress.

The entrance area was sculpted to create the appearance of an old tomb with an entrance pyre. These assets were also designed to be somewhat modular, and were repurposed for the parallax

pieces that decorate the distant background of each area. Some free Qixel Megascans assets from the Unreal Library[11] were added for small plants, foliage, and ground cover. To view primary sidescroller level assets, see Appendix B.

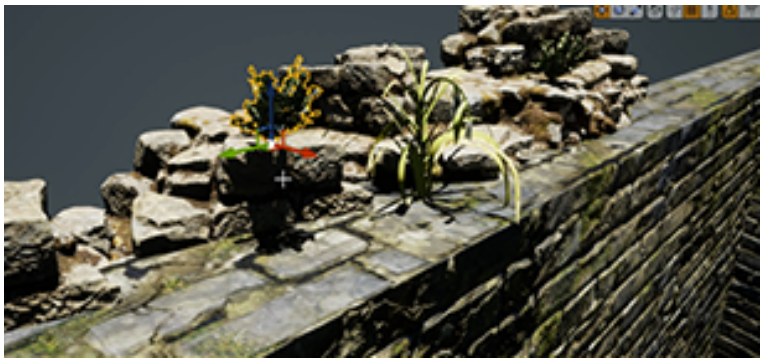


Figure 4.14:
Set Dressing with Megascans

4.3 Level Assets - Secondary

Not every level in the game needs to be a vertical jump trial, so another level was added early on as a safe place of respite for players to return to any time. Similar to the camp levels in *Shovel Knight*, it's a small area designed to be a calming break from the challenges elsewhere in the game. It is also designed to be visited early on in the game so that the player may discover an item, such as the sword or some later addition to the Knight's inventory.

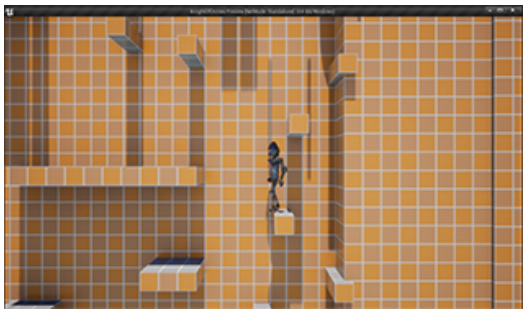


Figure 4.13: Level Blockout

This area, called the Knight's Guarden, is a green oasis with old rusted buckets of past battles scattering the landscape. A primary background asset starting as a quick sketch on scrap paper was sculpted in VR using Gravity Sketch, and was then exported and brought into ZBrush, where some cleanup was necessary. Each piece of the Gravity Sketch sculpt was ZRemeshed into cleaner quads, divided, and detailed using alphas. The high-res sculpt was then sent to Houdini via a GoZ port, where the model was decimated down, voxelized, and auto-UV'ed. This geometry was then brought over to Substance Painter for texturing.

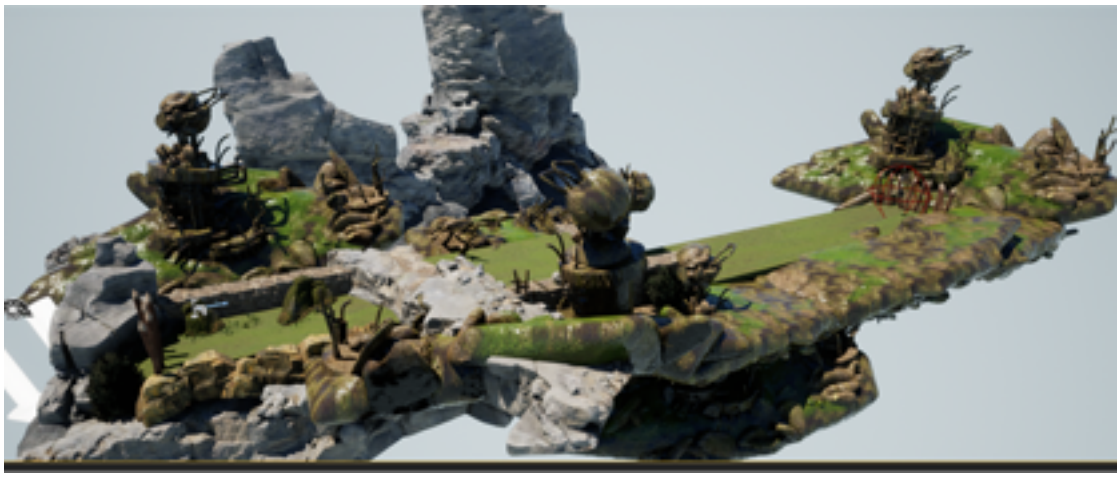


Figure 4.15:
Level Zero: Knight's Guarden

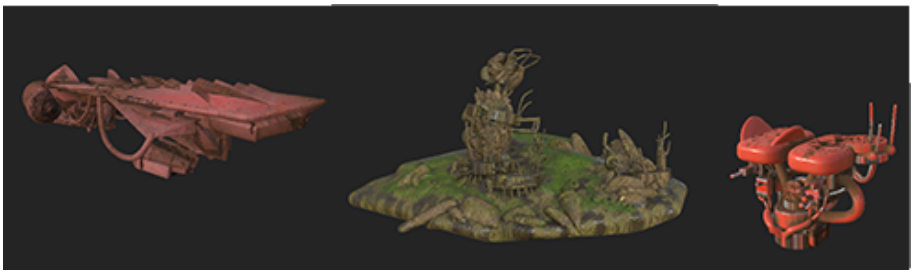


Figure 4.16:
Mothership, Knight's Guarden Background, and Turret Assets

The Gravity Sketch workflow combined with the auto-remesh and auto-UV functions of Houdini Side-FX Labs proved incredibly fast for non-hero assets, and this workflow was repeated for the mothership stand-in and first enemy turret assets that the player encounters in the top-down shooter portion of the game.

4.4 Overworld

The overworld is a large play space and the first area the player will encounter. The player explores the area to uncover new points of entry that lead to sidescroller maps. The main gameplay loop involves a mothership, which must be protected by the player. In the future, the player may return to the mothership for additional mechanics, such as restock ammunition and refuel. But for now, it represents the "castle," the home and origin point of the knight, who must protect it.

Several terrain tests using World Creator 2 (WC2) proved successful for generating assets with an aesthetically pleasing eroded landscape feel. These assets were used to generate height and flow maps that could be sent to Unreal Engine. However, during playtesting it became apparent that the height range needed to create naturalistic eroded terrain was too extreme. These terrains would be well suited to open areas in an RPG, where traversal of mountains and hills might create a sense of adventure and break up gameplay by occluding areas that would reveal themselves as the player successfully climbed over the top of a mountain. Because this is a top-down shooter, the height field needs to be relatively constant, broken up by spires and canyons that would cause the flying drone player to collide or crash. The gradual shifts in hills of the WC2 generated terrain look good, but in practice would alter gameplay so that the player's drone would not be constrained at a certain height. It was determined that the height available for gameplay would need to be quite limited as the drone needs to be able to hover fairly close to the ground to allow the player to eject and walk along the surface to enter new areas, as well as to constrain the player to a fixed height on the map. The height fields, when scaled in the Z (up/down) axis to squash them into a playable space, proved too gradual a change in the other axes and was not pleasant to the eye, so the height constraint in gameplay led to the abandonment of World Creator 2 as a terrain generator for this project.

Figure 4.17: Terrain Test

The material assets previously generated were combined with Epic Megagrant Recipient Joe Garth's[17] Brushify Canyon Biome asset to manually sculpt and paint the height fields of the map. This combined with 3D meshes copied and reoriented around the scene laid a strong foundation for the overworld map. Due to the camera's distance from the ground and the heavy texture load of the game's current assets, procedural grasses have been deactivated from the Brushify landscape material. Two additional materials were added to the Brushify material function to break up the natural materials with plastics, and erosion brushes were used on top of the sculpted terrain height field to instill some natural curvature. The height differences must remain steep, though, as the flying player controller must not skirt along the ground, and the secondary player controller must

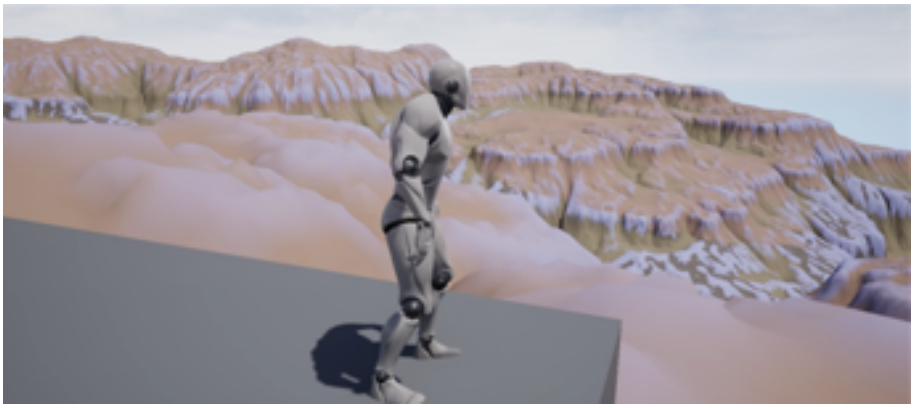


Figure 4.18:
Early World Creator 2 Height Field with Heat Maps in UE4

always have the ability to return to the position from which they have ejected. Because of these steep terrain changes, additional meshes will be needed to cover the texture stretching as anisotropy at this texture scale would be too expensive and potentially crater the framerate. More time must be spent carefully layering and painting the areas around level entries to help guide the player to important areas that load new levels.

Figure 4.19:
Brushify Assets Combined with Original Materials

The overworld continues to present a problem of scale. The sizes of pawns and player character meshes likely needs to increase, and the sizes of the turrets and motherships may need to decrease. At the same time, the purpose of the overworld is to become a fast-paced shooter with dodge mechanics. The player needs room to move and they need large bodies to maneuver around

when seeking cover. Scaling of gameplay elements needs to continue as combat is implemented and as additional playtesting occurs.

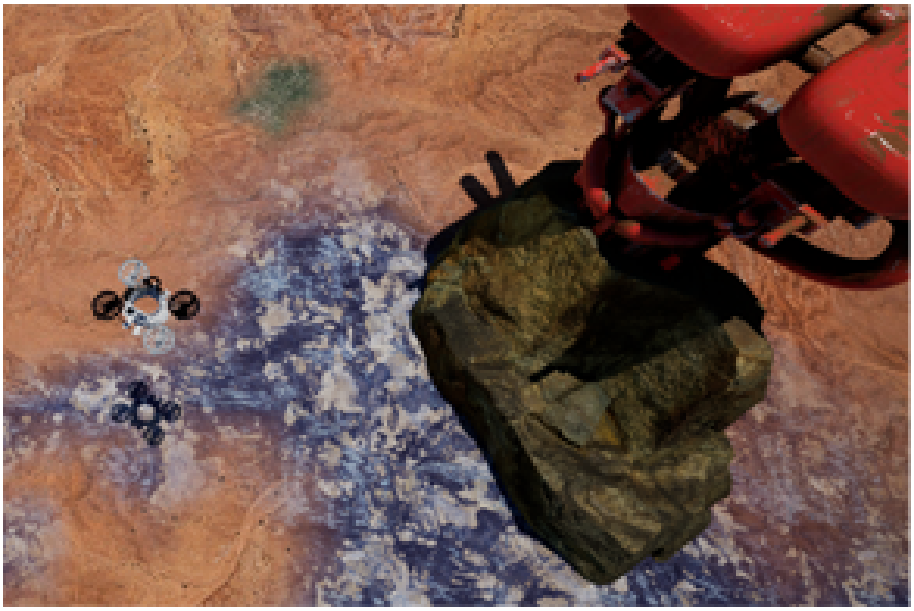


Figure 4.20:
Top-Down Shooter View of Overworld

4.5 Frame

The frame is the drone asset for the top-down shooter and has been through several phases of sketches. Currently the asset is a stand-in box model from Maya with two pieces: a body and a turret. The body is designed to take fans as a quad copter drone would, and the turret is a two-barrel machine gun design.

The model was lit and rendered with Arnold to provide source images for photobashing and speedpainting concepts for further development of the frame. The frame holds the knight, and thus should be considered a vehicle more than a familiar or steed, and it is left on its own as the player switches from the frame class to the mini-knight class in the top-down overworld. The frame needs more design iterations before a final model is attempted; for now, the stand-in asset has served well to build the basic shooter mechanics.

Future frame iterations of design need to be built around the abilities of the player, thus more playtesting and mechanics design needs to occur before further visual design of this asset

should take place. Known mechanics that need to be developed and tested include a boost function and a secondary weapon that can lock on to nearby targets and fire; this mechanic will need a design element on the frame that will animate. The current frame is a pawn using two static meshes. It is likely that the final version of the frame will need a skeletal mesh to take some simple animations for different weapons firing, etc.

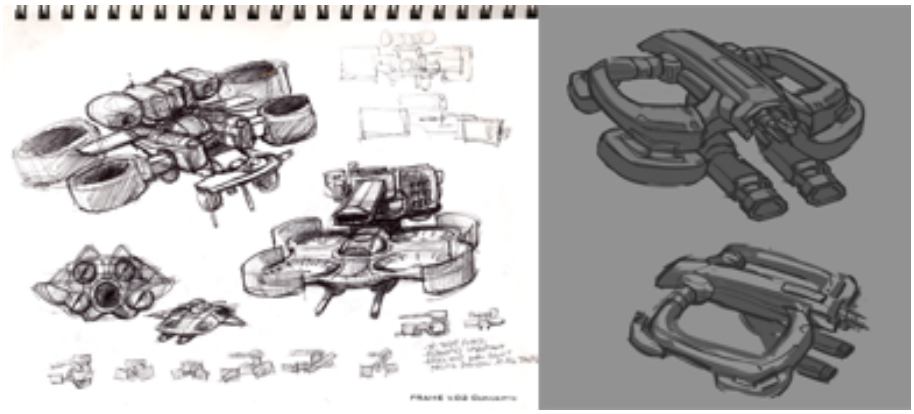


Figure 4.22:
Frame Concepts Continued

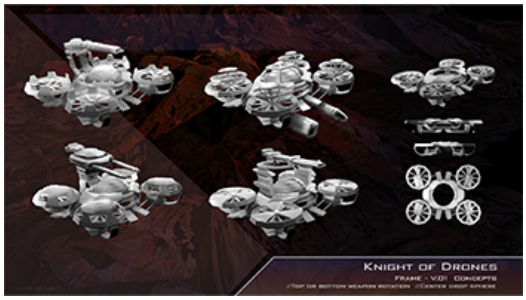


Figure 4.21: Frame Concept Overpaint

Chapter 5

Design: Gameplay Mechanics

5.1 Implementation and Gameplay Methodology

Knight of Drones is built in Unreal Engine 4 (UE4) and features three different player controllers spanning two modes of gameplay. All player controllers have defined inputs in the engine's project settings that route key presses and joystick movement to the proper functions. Each player controller has a collider and some method of binding input from the player to movement or other kinds of feedback. The preferred controller for Knight of Drones is a gamepad with two analog sticks. All game classes are built in C++ and later used as source code for blueprints, which share similarities to 'prefabs' in Unity. Blueprints can also refer to the visual scripting system built into Unreal Engine; this can confuse new users at first. A blueprint may refer to a collection of game objects, such as a player controller with an animated mesh or a wall sconce with a point light and a fiery particle system. These types of blueprints may often be dragged into the map and placed in the game world. In this way, 'blueprint' is a catch-all term for many types of game objects. However, blueprints is also the node-based visual scripting system in UE4, which can be used to control event flow or create functions and variables. Coding in C++ is generally more modular and versatile while blueprint scripting is considered faster and more accessible. While it is possible to create an entire game using only blueprints (the scripting platform), it is generally accepted as good practice to use both C++ and blueprints in UE4. Blueprints can become cumbersome and difficult to read over time, and thus their use is not ideal for collaboration. C++ can occasionally cause a derived blueprint to stop updating, which forces the user to save their work and close the project to rebuild

the Visual Studio files. Knowing these parts of the UE4 workflow ahead of time can be helpful in making decisions about which tools to use when.

The most important gameplay mechanics start with the player controllers. The player uses these to interact with the game world. The player controllers that receive input drive the avatars or the interfaces through which the player experiences the game.

The top-down shooter controller in Knight of Drones, the Frame, inherits from the pawn class. Pawns are any actor class that can be possessed by a user or an AI. The Frame is constrained in the Z axis (Z is 'up' in UE4) so that the player may only move on the X and Y axis. In addition to a mesh and the player movement component, a camera actor attached to a springarm component, a functioning camera boom, attaches to the pawn's root component, in this case a capsule collider. The springarm has a built in camera drag function so that the camera drifts behind slightly, and this helps give a sense of weight and speed to the player's movement. The Frame also has an assortment of attributes and functions, such as move speed and the ability to shoot(), which is a function that spawns a projectile class that contains a collider with speed and direction. As is common with many moving actors, the Frame also has getters and setters for location, which are crucial for moving the pawn through space or spawning a projectile from the current mesh location. The Frame also has a slight up and down drift applied to the Z axis during runtime to give it a feel of floating in air. This function is built with a simple SIN function from Unreal's FMATH library. When dealing with a speedy pawn constrained in the air, it's best not to look like a wet bar of soap sliding along a countertop. This sine wave function creates an illusion of in-air hovering for the Frame actor.

The Frame class is also the source of a pawn swap. The player may eject from the Frame at any time to drop a new controller onto the ground. This new controller is of the character class, which derives from the pawn class. Character classes in UE4 are built to take input and have default movement modes like walking, swimming, and flying. The character class that drops out of the Frame is called the MiniKnight(MK), a tiny version of the early version 01 Knight. This small-scale character class only needs some limited functionality such as input that translates the actor so that the player may walk along terrain, and the MK class also includes an animation blueprint and a collider to detect when the actor is overlapping with other game objects. This MK class provides the method by which players overlap with colliders on the overworld terrain in order to load a side scroller level. At the moment, there isn't much need for additional functionality to the MiniKnight, but future iterations might include a run/sprint function or a sword swing attack() to knock aside

barriers at ground level in the overworld. The pawn swap is not implemented in the pawn class as it involves multiple interacting classes (pawn and character classes). See later in this section for more on the character controller swap.

The most complex player controller class is the side scroller player controller, the Knight. The Knight character takes a skeletal mesh component with several animation loops. It needs a collider, of course, and it also features a socket at the center of the right hand wrist joint to which an item can be equipped. The sword is a static mesh and is equipped to the the Knight class via an overlap item pickup. The sword itself is an Item class; An Item is a modified Actor class that comes with built-in overlap checks and a simple getter/setter pair for overlaps. This getter/setter pair helps with any future item overlap, be it item collection or a weapon striking an enemy. The Knight player controller represents many hours of tweaking parameters to get the proper gameplay feel. Jump velocity, ground friction, air control, camera drag, running speed, and rotation speed (for when the character rotates to the left or right when changing direction) are several instances of parameters that required tight adjustment to fit the feel and flow of a vertical platformer. See the source code for Knight.h and Knight.cpp in Appendix D. The Knight character class also takes an animation instance, which is a very simple C++ class that may be sourced to create the animation blueprint. This animation blueprint takes the segmented animation clips as inputs for the animation state machine and sends the necessary signals from the animation frames to the skeletal mesh based on the rules of the state machine.

Despite the flexibility of C++, ready and easy access to multiple classes are available in the game mode and level blueprints. The terms 'level' and 'map' are used interchangeably hereafter and refer to an Unreal Engine asset that contains a playable area of the game into which player controllers may be spawned. Levels may contain their own rulesets and assets, and in this case different levels will load different player controllers. UE4 base game modes define a default pawn class as a controller to spawn when the game begins. Due to the player swapping controllers during gameplay, it was necessary to disable the default pawn and to define a pawn possession at the start of gameplay depending on which level was loaded. Therefore the primary method that makes pawn swapping possible is the level blueprint, although this could also potentially be relegated to the game mode blueprint. Because the sidescroller levels would need to load a separate camera and set of assets at a different world scale, level blueprints were selected as the location to perform player controller swapping.

When the game starts, the player spawns in the overworld as a Frame pawn. This takes the defined top-down shooter parameters. The player may then press the B button on their controller (or the E key) to initiate the following: The player controller dispossesses the frame pawn, a MiniKnight character spawns at the location of the center of the frame, the player possesses the MK class, and the MiniKnight responds to gravity and thus falls to the ground. Once walking, the MiniKnight character is free to roam. From here, it's necessary to provide the player with the ability to return to the Frame class to continue flying and exploring. This is achieved with the same B Button / E key input, which causes the MK class to cast a ray upwards to perform a collision check. If the ray collides with the collision mesh of the Frame class, the MiniKnight depossesses, destructs, and the player repossesses the Frame pawn to continue flying about. As the MiniKnight class, the player may also choose to explore the ground plane of the overworld level in search of entrances to other levels. Once the MK class overlaps with an entrance, a level load is called. With the level load, a Knight class is constructed and possessed by the player and the sidescroller portion of the game is in play.

5.2 Iteration

Initial builds of the pawn swapping mechanic were built into the pawn and character classes. These proved unsuccessful. Immediate problems became apparent when class casting needed to occur between classes, some that hadn't even yet been constructed. Pawn swapping mechanics are quite common in Unreal Engine, particularly during multiplayer games where player deaths and respawns occur. This highly functional mechanic seems built for server-level commands, game modes, and levels. Thankfully the possess and dispossess functions are readily available for use because of these conventions.

At the start of play, the overworld level executes the first player possession call. The Frame class is spawned and the player automatically possesses this pawn. The Frame class blueprint contains an event bound to a key press (the B-button or E key on the keyboard) that constructs a MiniKnight class at the position of the Frame. Gravity drops the MK to the terrain and a blueprint call depossesses the Frame and possesses the MK class. As soon as the MK class touches the ground, the player is free to move using the MK player controller. From here they may explore the overworld on foot or choose to return to the Frame. The MK class has an event call for the action button B

(or E key) that casts an upward ray check. If the ray collides with the Frame mesh component, the MK class destructs and the player once again possesses the Frame class. To view the blueprints for pawn swapping functionality, see Appendix C. The player also needs feedback as to the position of the Frame as it stands idle while they walk around with the MK class. Currently, the Light Studio blueprint of the overworld has a directional light in the scene set to shine directly down onto the terrain, which casts a shadow directly underneath the Frame mesh. The player may use this as a destination marker to return to the Frame. In the future, a decal or visual effect can be added as the MK class drops to the ground as an indication of where the player must return to to initiate repossession of the Frame.

The side-scroller levels are a fresh level load called by the MK class overlapping with a box collider placed on the terrain of the overworld. A level load creates a pause on many systems while the engine shifts geometry and texture data. As Knight of Drones continues to develop, attention should be given to the transitions between the top-down map and the sidescroller levels. A simple Linear Interpolated (LERP'ed) camera move through parallax art assets to align to the side-scroller player camera could be an artistically effective way to create an in-engine 'wipe' to reveal the new level.

Pawn swapping and level loading iterations have been tested and proven successful. A somewhat foreseen but unresolved mechanic that needs several more builds is a behind-the-scenes autosave system that preserves the states between levels. Typical save states involve specifically selected data points that are written to save files for later recall. An inventory is easy to imagine; a player carries 152 coins. They save their game. They should load the game and find their pouch packed with 152 coins. In this same way, one could extrapolate that an idle pawn class needs to set its position data to a save file for later recall. The complication becomes that setting a pawn's location is different than saving its possession state. If a level's default loading pawn is a Frame, and the save file loads a MiniKnight and a Frame in two positions, the level blueprint will override save function and automatically possess the level pawn.

The most important progression for the game is to iterate on the save state functions that will allow the player to seamlessly hop between levels keeping their position, inventory, and progression intact. A technical and artistic hunch points towards the game mode blueprint, which serves as the director of gameplay and the event coordinator and chaperone. This is more important than enemy or friendly NPC's, item or experience based progression, or shooter mechanics. Without the ability

to link the sidescroller levels and overworld map together, there is no game. The save/load mechanic that will link the level states together is the immediate, primary concern for the state of the game.

Chapter 6

Results

The majority of endeavors for each facet of the project have been successful. Importantly, the gameplay is fun. This is a good sign, and a good indicator that development should continue.

Several additional mechanics need to be in place before the game reaches an alpha state. These include an automatic save/load function that serves to link the top-down and side-scroller levels together, a mothership health function that leads to a game over state upon depletion, Knight player vs. NPC combat mechanics, and Frame (top-down) shooter game mode mechanics.

Figure 6.1:
Animation Cycles in Unreal Engine

The most important aspect of gameplay, the player controllers, are in place and working well. The animation cycles need adjustment. In the case of the run cycle, a better loop is necessary. The attack cycle needs an improved backwards arc for the anticipation and a stronger swing forward, possibly with a horizontal slash and a vertical slice. The in-air cycle for falls and drops needs to read better in silhouette, and the addition of an in-air attack swing would expand the player controller in

to more playable and fun territory. Additionally a duck and duck attack should be added for variety of attack response to NPC enemies. The FK rig of the knight was found to be serviceable, but an IK/FK switch rig, or even better, a rig with rubberbones to allow for comical squash and stretch with even the most mechanical of elements could prove beneficial for a game that needs a veneer of cheek to become successful.

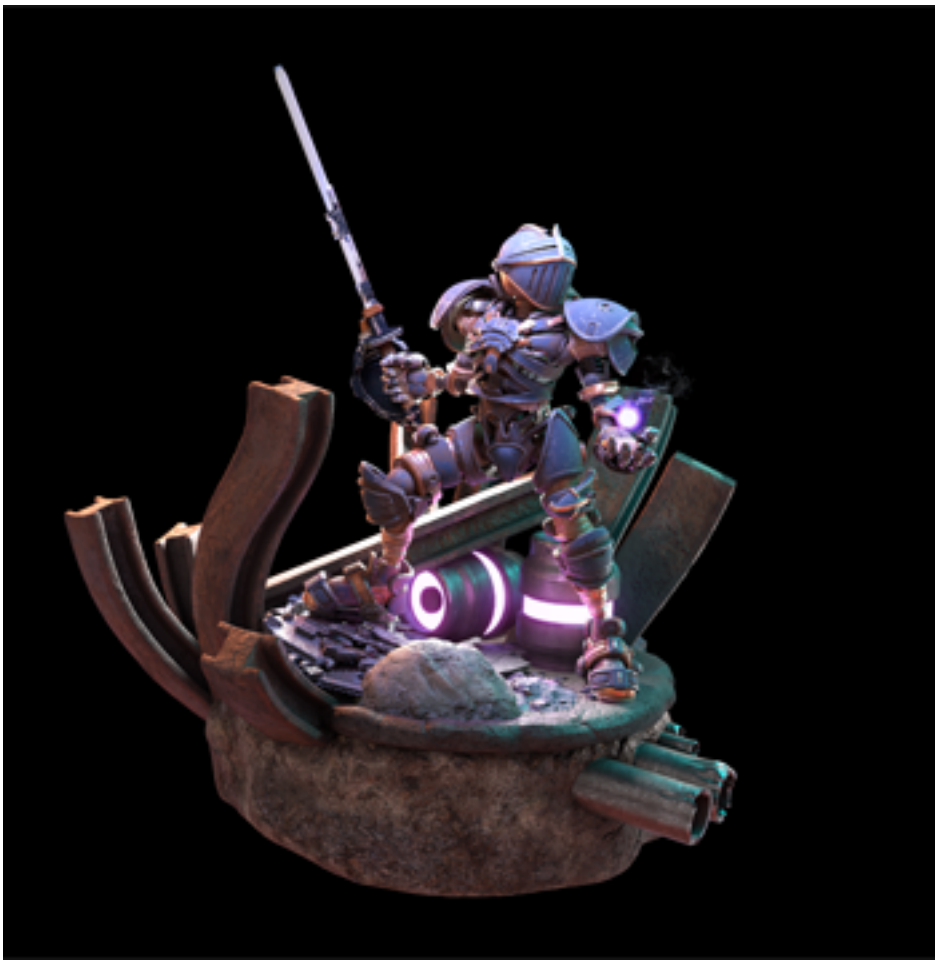


Figure 6.2:
Knight Final Render: Textured and Posed on a Base Sculpt

The Knight character stands out as a success of design, modeling, texturing, and rendering. The head and face highlight a feature set where additional effort could improve feedback to the player and increase personality and appeal. However, overall, the creative and technical pipelines unified to create a piece of work that both aesthetically intrigues and technically performs. It looks great, and it shipped to a game engine as an interactive asset. In this way, the Knight is a highly

successful project in itself. The game resolution asset, lit and rendered with Arnold renderer in Maya, captures the look and feel of Knight of Drones in a single image.

Figure 6.3:
Knight's Guarden: Successful Level Build

The side scroller levels represent successful level builds. Colliders have been placed to prevent the character controller from disorienting from the plane constraint and falling off the map. In the event of a player controller becoming disoriented, a kill box collider at the bottom of each level is ready to take reset or kill commands. For now, the box trigger ends gameplay.

The pawn swapping experiment succeeded completely where applied, and needed additional research and implementation in areas of admitted blind spots, namely the save/load function necessary for proper recall of game states between level loads. The game currently lacks NPC enemies and friendly NPC encounters, but these should be logged in the TODO list and not in the failure-to-have implemented list. Indeed, a carefully designed player controller is more important than an enemy spawn volume.

The game also needs a HUD for player feedback, and clear death states or respawn states. Additional time must be spent on optimization as well. In its current form, the game is slightly heavy in the area of texture memory.

As mentioned earlier, the scale problem of the overworld should be considered a design concern of note, and one whose solution could lead to the ultimate success or failure of the game. If the top-down shooter camera doesn't reveal enough or adjust its zoom or placement with the number of adversaries and events onscreen, frustration will result and players will move on. Along similar lines, the color choices and separation between the player characters and background assets must be enhanced for clarity of image separation. Even in dark areas or saturated desert sands, the player must know at a glance their avatar's position onscreen.

Chapter 7

Conclusions and Discussion

7.1 Conclusions

A number of important lessons of technical and artistic process were learned throughout the course of this project. Implementation of game design mechanics was done as early as possible and engine grayboxing took place well before artistic asset generation began. Still as the player controllers were designed, it became increasingly apparent that game mechanics and dynamics will shift and change in ways that can impact artistic assets. The Frame design was one of the first elements to have concepting iteration and remains one of the final assets to be completed. More details must be known regarding the full needs of the Frame class and how it affects the flow of the game before designs begin to harden into final game assets. Through playtesting with the current Frame asset, it was discovered early on that the player would need to see forward facing direction clearly, but further testing revealed that the model must be asymmetrical and have a stronger silhouette. This may clash with the recognizable quad-copter drone silhouette as these are commonly symmetrical. Additionally, a two-part mesh is insufficient as later functions will require visual feedback.

The complexity of the hard surface Knight sculpt provided excellent practice blending mechanical design with medieval appearance. This work represents both significant effort towards building a functioning hero asset for an independent game project and a solid source for what must be designed for the eventual Frame asset. Larger moving parts and a modular structure with multiple sockets are defining features of the next iteration of the Frame asset so that it can take various skeletal mesh assets that animate when aiming and firing. The Frame must also be designed with

rotors that can move and animate depending on the movement input of the player, so more research must done on real-life drone operation.

Having a clear, focused design missive or style bible at the start of a project helps guide the look and colors of the game through the production process. A game design document was prepared as a preproduction asset to help organize classes and systems as the playable prototype progressed, which was a valuable tool. More concept art and more stylistic exploration would serve the project well by establishing color, lighting, and thematic tones. These need to be more clearly established in the game and gameplay moving forward so that the narrative themes of the game do not get lost during design and implementation. Singular assets and level art pieces in this prototype are successful and serve their purposes, but increased numbers of assets and higher level complexity left several areas of artistic opportunity unexplored. There should be more coordination between the overworld level art areas and the side-scroller level areas, for instance, which can be achieved with more careful planning and greater attention to detail when moving forward with level design and modular asset creation. In short, Knight of Drones needs Art Direction. The decision to create PBR textures looks great in engine, especially with RTX raytracing enabled, but the color choices of aged dirty stone and metals introduces a washed-out, muddy look. The game needs more color, and the player controller assets need help standing out from the levels.

Balancing game design with art asset generation was a known difficulty, but even so, the cross-feedback between game design and visual design presented many unforeseen challenges. It is entirely possible to iterate on designs that do not comply with gameplay, for instance, or to find that an artistic asset, such as an animation, can work well during playback but not fit the needs of the game engine. It is also possible to make color choices that work well in an evening or night time setting that will not work well in a daytime scene, and so game lighting and design must also work with the time of the setting. More choices should be made around the game's day/night cycle, if there is to be one, and to make sure that assets are test-rendered using settings and lights that best represent the needs of the game.

These lessons represent a few of the highly valuable experiences that will inform decision making and design choices in future iterations of Knight of Drones. The game plays well, and has a long way to go before it resembles a full title for players to enjoy, but this working prototype successfully demonstrates the core concepts of the game and remains a clear example of dedication to the craft of game design and artistic discipline.

Appendices

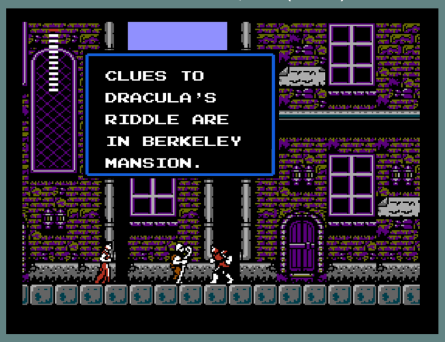
Appendix A Shovel Knight Mechanics Comparison

Super mario Bros. 3(1988)

Shovel Knight (2018)

Overworld map with player navigation, level selection, animated environment elements, and traveling miniboss encounters

Castlevania II: Simon's Quest(1988)



Shovel Knight (2018)

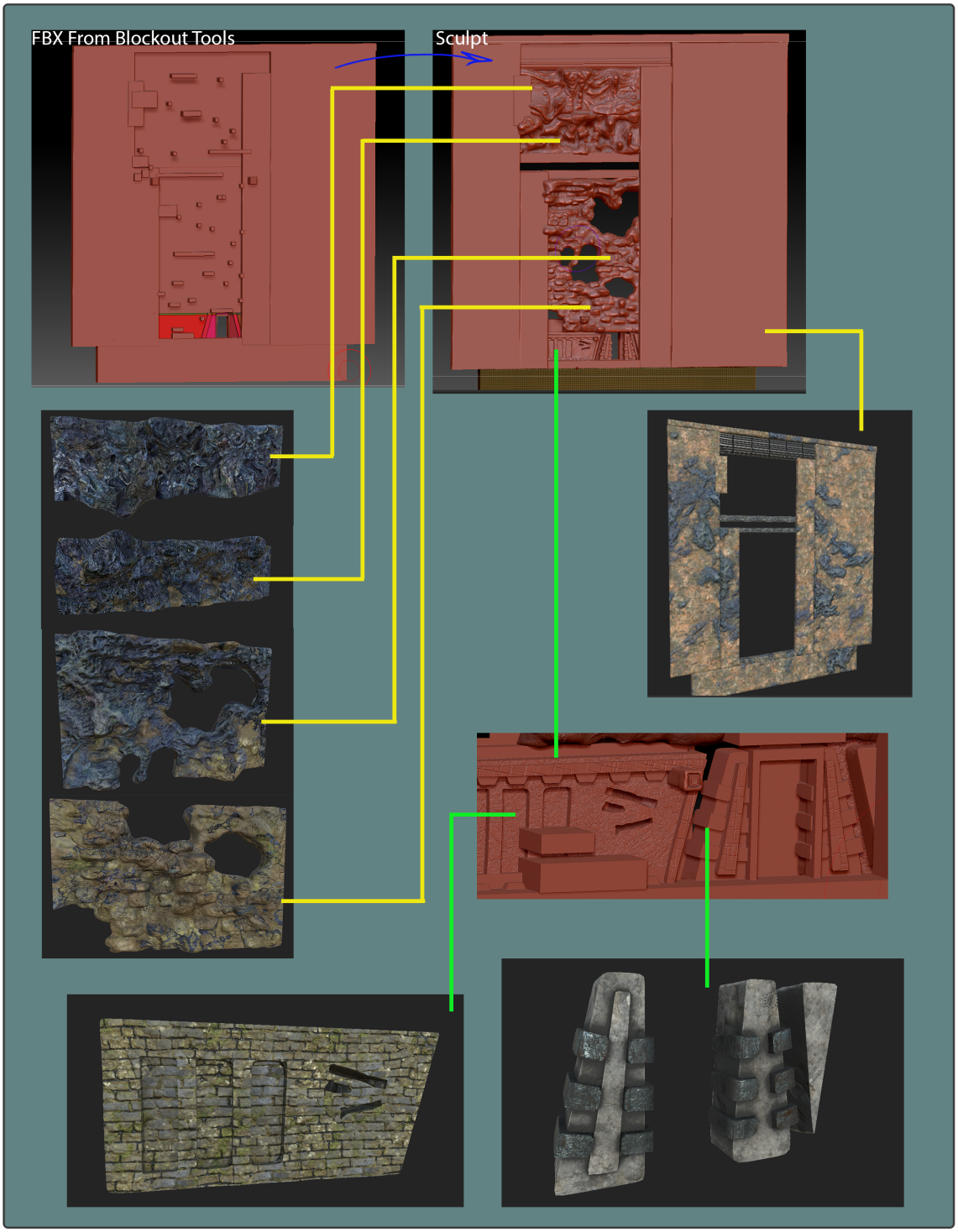
Wandering village NPCs, Internal and external map entry/exit, Secret areas

Duck Tales(1989)

Shovel Knight (2018)

Pogo drop attack and bounce mechanic, Static level asset swipe attack mechanic

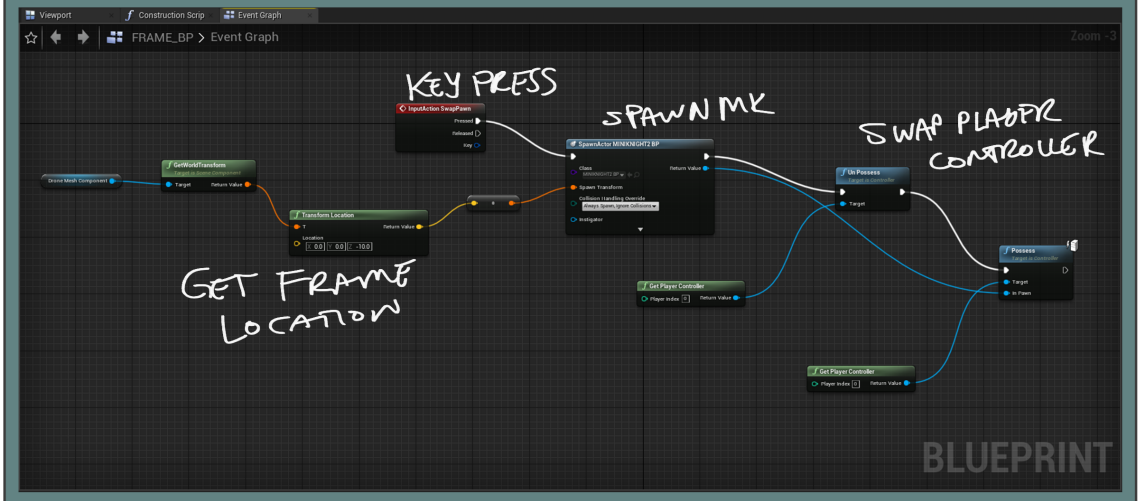
Appendix B Primary Side Scroller Level Asset Generation



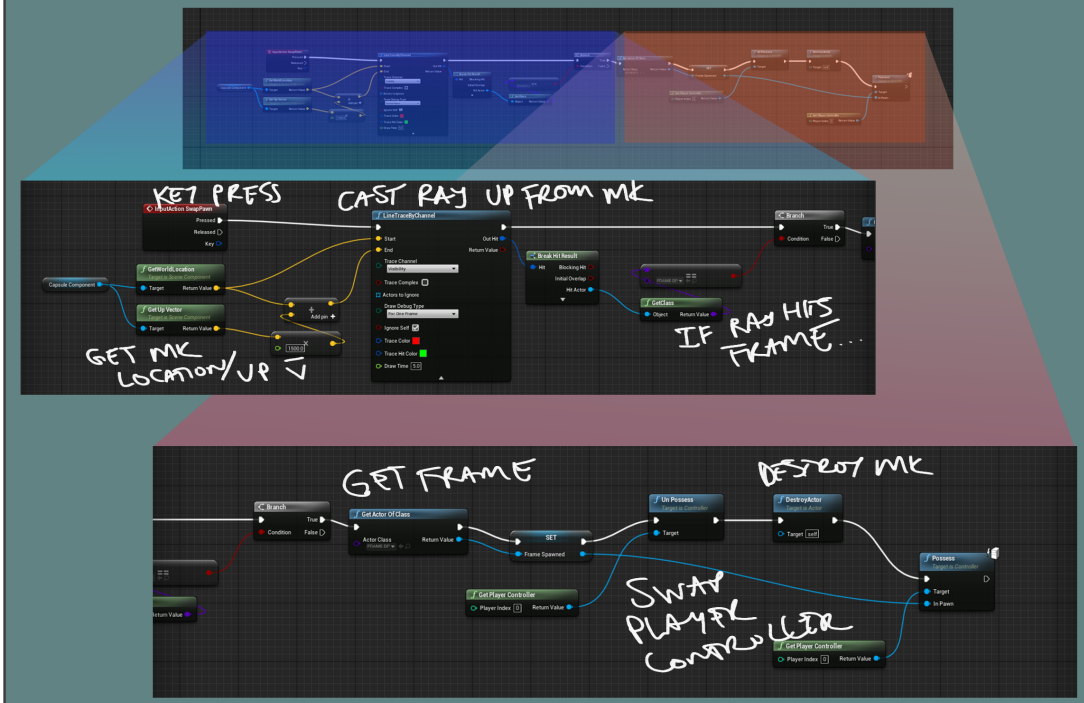
Brick Sculpts and Textures

Appendix C Pawn Swapping Blueprints

Swap from FRAME to MINIKNIGHT



Swap from MINIKNIGHT back to FRAME



Appendix D Knight Player Controller

Knight.h

```
// Fill out your copyright notice in the Description page of Project Settings.

#pragma once

#include "CoreMinimal.h"
#include "GameFramework/Character.h"
#include "KNIGHT.generated.h"

UCLASS()
class KNIGHTOFDRONES_API AKNIGHT : public ACharacter
{
  GENERATED_BODY()

public:
  // Sets default values for this character's properties
  AKNIGHT();



  /** Camera Boom positioning the camera to the side of the player */
  UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = Camera, Meta = (
    AllowPrivateAccess = "true"))
  class USpringArmComponent* CameraBoom;

  /** Camera that follows the player */
  UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = Camera, Meta = (
    AllowPrivateAccess = "true"))
  class UCameraComponent* FollowCamera;



protected:
  // Called when the game starts or when spawned
  virtual void BeginPlay() override;

  // Called to bind functionality to input
```

```
35   virtual void SetupPlayerInputComponent(class UInputComponent* PlayerInputComponent)
        override;
36
37   /** Called for side to side input */
38   void KnightMoveRight(float input);
39
40   /** Handle touch inputs. */
41   void TouchStarted(const ETouchIndex::Type FingerIndex, const FVector Location);
42
43   /** Handle touch stop event. */
44   void TouchStopped(const ETouchIndex::Type FingerIndex, const FVector Location);
45
46 public:
47   // Called every frame
48   virtual void Tick(float DeltaTime) override;
49
50   FORCEINLINE class USpringArmComponent* GetCameraBoom() const { return CameraBoom; }
51   FORCEINLINE class UCameraComponent* GetFollowCamera() const { return FollowCamera;
       }
52
53   UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, Category = Items)
54   class ASword* EquippedWeapon;
55
56   FORCEINLINE void SetEquippedWeapon(ASword* WeaponToSet) { EquippedWeapon =
       WeaponToSet; }
57
58 };
```

Knight.cpp

```
1  // Fill out your copyright notice in the Description page of Project Settings.
2
3
4  #include "KNIGHT.h"
5  #include "GameFramework/SpringArmComponent.h"
6  #include "Camera/CameraComponent.h"
7  #include "Engine/World.h"
8  #include "Components/CapsuleComponent.h"
9  #include "GameFramework/CharacterMovementComponent.h"
10
```

```
// Sets default values
AKNIGHT::AKNIGHT()
{
  // Set this character to call Tick() every frame.  You can turn this off to improve
     performance if you don't need it.
  PrimaryActorTick.bCanEverTick = true;

  //Set size for collision capsule
  GetCapsuleComponent()->SetCapsuleSize(35.f, 105.f);

  // Don't rotate when the controller rotates.
  bUseControllerRotationPitch = false;
  bUseControllerRotationYaw = false;
  bUseControllerRotationRoll = false;

  // Create Camera Boom (pulls twds player if there's a collision)
  CameraBoom = CreateDefaultSubobject<USpringArmComponent>(TEXT("CameraBoom"));
  CameraBoom->SetupAttachment(GetRootComponent());
  CameraBoom->SetUsingAbsoluteRotation(true); // Rotation of the character should not
     affect rotation of boom
  CameraBoom->bDoCollisionTest = false;
  CameraBoom->TargetArmLength = 420.f; //Camera follows at this distance
  CameraBoom->bUsePawnControlRotation = false; //Do not rotate spring arm based on
    the controller's rotation
  CameraBoom->SocketOffset = FVector(0.f, 0.f, 75.f);
  CameraBoom->SetRelativeRotation(FRotator(0.f, -180.f, 0.f));
  CameraBoom->bEnableCameraLag = true;
  CameraBoom->CameraLagMaxDistance = 200.f;
  CameraBoom->CameraLagSpeed = 3.0f;

  //Create camera that will follow player and attach it to the boom
  FollowCamera = CreateDefaultSubobject<UCameraComponent>(TEXT("FollowCamera"));
  FollowCamera->SetupAttachment(CameraBoom, USpringArmComponent::SocketName);
  FollowCamera->bUsePawnControlRotation = false;

  //Characer movement components
  //and constrain to plane!
  //GetCharacterMovement()->JumpZVelocity = 450.f;
  //GetCharacterMovement()->AirControl = 0.2f;
```

```
47    //GetCharacterMovement()->bConstrainToPlane = true;
48    //GetCharacterMovement()->SetPlaneConstraintNormal(FVector(1.f, 0.f, 0.f));
49    //Fixed collision issues drifting player with vectors instead of normal
50    //Constrain to Y,Z axes
51    //GetCharacterMovement()->SetPlaneConstraintFromVectors(FVector(0.f, 1.f, 0.f),
        FVector(0.f, 0.f, 1.f));
52
53    GetCharacterMovement()->bOrientRotationToMovement = true; // Face in the direction
        we are moving..
54    GetCharacterMovement()->RotationRate = FRotator(0.0f, 720.0f, 0.0f); // ...at this
        rotation rate
55    GetCharacterMovement()->GravityScale = 2.f;
56    GetCharacterMovement()->AirControl = 0.80f;
57    GetCharacterMovement()->JumpZVelocity = 1200.f;
58    GetCharacterMovement()->GroundFriction = 3.f;
59    GetCharacterMovement()->MaxWalkSpeed = 600.f;
60    GetCharacterMovement()->MaxFlySpeed = 600.f;
61
62    // Note: The skeletal mesh and anim blueprint references on the Mesh component (
        inherited from Character)
63    // are set in the derived blueprint asset named MyCharacter (to avoid direct
        content references in C++)
64
65
66    //TODO: Constrain player X so theres no drift from collisions
67
68
69  }
70
71  // Called when the game starts or when spawned
72  void AKNIGHT::BeginPlay()
73  {
74    Super::BeginPlay();
75
76
77
78  }
79
80  // Called every frame
```

```
81 void AKNIGHT::Tick(float DeltaTime)
82 {
83   Super::Tick(DeltaTime);
84 
85 }
86 
87 // Called to bind functionality to input
88 void AKNIGHT::SetupPlayerInputComponent(UInputComponent* PlayerInputComponent)
89 {
90   Super::SetupPlayerInputComponent(PlayerInputComponent);
91   check(PlayerInputComponent);
92 
93 
94   //Character includes a jump function already, so qualify character class
95   PlayerInputComponent->BindAction("Jump", IE_Pressed, this, &ACharacter::Jump);
96   PlayerInputComponent->BindAction("Jump", IE_Released, this, &ACharacter::
       StopJumping);
97 
98   //PlayerInputComponent->BindAxis("MoveForward", this, &AKNIGHT::MoveForward);
99   PlayerInputComponent->BindAxis("KnightMoveRight", this, &AKNIGHT::KnightMoveRight);
100 
101   PlayerInputComponent->BindTouch(IE_Pressed, this, &AKNIGHT::TouchStarted);
102   PlayerInputComponent->BindTouch(IE_Released, this, &AKNIGHT::TouchStopped);
103 
104 }
105 
106 void AKNIGHT::KnightMoveRight(float input)
107 {
108   if ((Controller != nullptr) && (input != 0.0f))
109   {
110 
111     AddMovementInput(FVector(0.f, -1.f, 0.f), input);
112 
113   }
114 
115 }
116 
117 void AKNIGHT::TouchStarted(const ETouchIndex::Type FingerIndex, const FVector
       Location)
```

```
118 {
119   // jump on any touch
120   Jump();
121 }
122
123 void AKNIGHT::TouchStopped(const ETouchIndex::Type FingerIndex, const FVector
        Location)
124 {
125   StopJumping();
126 }
```

Appendix E Game Design Document

Martell DPA Thesis
Knight of Drones Game Document

Title: Knight of Drones
Engine: UE4
Genre: Action/Platformer

Description

Knight of Drones is a tongue-in-cheek top-down shooter/2d platformer hybrid with light strategy mechanics.

The look is a mashup of post-apocalypse, technology, and medievalism. The game combines many simple mechanics together and draws heavy inspiration from 8-bit era vintage games.

Story

Knight of Drones takes place in a post-human world where all that remains is the autonomous/unmanned war machines programmed for battle. Players take control of a flying drone, called the Frame, and a robotic protagonist, the Knight.

Players awaken as a unique AI function that can hop between operating a top-down shooter drone and a side-scrolling platformer character. Initially, the game is a straightforward action game where players win battles to progress. The narrative builds tension as the gameplay increases in difficulty. Midgame or late-game, it's revealed that humans still exist underground and the player can choose to stop destroying their facilities and make some allies, or choose to continue to fight on the side of the machines and eradicate the last bastion of humanity.

The story should be told with gameplay and, if necessary, pictograms of gameplay elements. If/when the game gets finished, translation and localization to other territories will be minimal- all that needs translation is menus.

Gameplay

The FRAME

The primary game mode of the main map, called the Overworld, is a top-down twin-stick shooter where the player controls a quadcopter drone called the FRAME. It hovers at a constant height and can move (L stick) and fire (R stick) in a circle at its own height.

Future actions could include dropping bombs or grenades (x button) onto the ground. These projectiles could have different functions and ranges (some explode out, some up, some dig, some are EMP, etc). R-stick firing is unlimited ammo, grenades/bombs are limited and need to be refilled.

The Frame can also be upgraded to dodge short distances (y button), use a radar/magnetic ping (b button) to reveal or pinpoint areas of interest.

Static enemies (turrets, etc) litter the world and need to be cleared to progress. Enemy drone NPCs swarm and attack much of the time. Static spawn points throw these enemy drones when player is within a certain distance. Players destroy spawn points to get some respite and also to progress.

The player can also eject the entity that's being carried in the center of the frame, which drops to the ground and unfolds to become:

The MINIKNIGHT

Now the player is a tiny knight walking on the terrain of the game world. The mini knight on the ground of the overworld is vulnerable to attack and later random/scripted events. But the purpose of ejecting down to the ground is to walk into entrances to caves, mines, buildings, castles, etc. Once inside the game/camera transitions to a sidescrolling game.

The KNIGHT

Once the side-scroller level has loaded, the player assumes control of the Knight.

Inspired by Rare's early NES game *Wizards and Warriors*, it's a vertically oriented platformer for the most part. Players climb tall heights or fall long depths. But much like *Wizards & Warriors*, jumping is the best attack and character placement on platforms is a strategy in itself.

The KNIGHT has a thin sword like a rapier. Great for parrying and slapping but not ideal for ground combat since the player will mostly be worried about jumping.

The side-scrolling levels will be lots of jumping and dodging, but will have some tough enemies that require slapping an enemy at a weak spot, and a few bosses as things progress. Bosses will likely need to be jumped over (hitting them with the sword on the way over).

The KNIGHT will encounter some switches or light puzzles that open things in the overworld outside. Then the KNIGHT goes back outside to the overworld and hops back into the FRAME (if it's still there or there isn't a story event, etc.) and heads back home to home base to refuel and refill ammo. Home is:

The MOTHERSHIP

There's a giant hovering drone that the player starts at, and every so often the player returns or spawns back at this big ship. Here there are 3 friendly NPC's: the Commander, The Shopkeep, and the Intel bot. They're all bots and all of them just make noises and show pictograms to communicate anything to the player.

The mothership has little moments to progress story and it takes in resources the player brings home.

The ultimate goal of the game is to move the mothership to the final place on the map, the Commander NPC will overtake it and all the enemy bots in the territory will come under Commander's control. The mothership only moves via a large pause map, and it moves very slowly. The player needs to clear turrets and spawn points on the overworld to make the world safe for the mothership to move. This could also open up a defense mechanic where the player must rush back to the mothership to protect it from some randomly spawned waves of enemies, but this mechanic must be used carefully/sparingly because the flow of the game must not be interrupted. There may be room here for a mechanic for the player to spend resources to add/build defensive turret additions to the mothership. These could increase its speed, have an auto-fire defense turret, etc. This adds more direct need for resources and takes the pressure off the player mid-game, allowing them to explore farther without worrying about incoming attacks to the mothership.

This is the strategy element and the mothership also serves as a respawn point for the player and a game-over state if the mothership takes too much damage and is lost.

It also could introduce a fun story element where if the player decides to save the humans later in the story, the humans start to show up at the mothership and cause messes/problems in the background. Like while the player is shopping a big galoot pops into the shop and knocks a bunch of items over bothering the Shopkeep bot, etc.

Music / Audio

Building the audio system in Unreal Engine is an area of interest, but may be outside the scope of this prototype.

The music should be a fun mix of medieval/Baroque organ/harpsichord riffs and german booty bass/electro. It's not strictly analog in sound and the music should be separated into stems that build or drop as the player is performing actions. Intense music during combat will shift through reverb and drift into calm ambiance during pause screen, then ramp back up after pause menu exits, etc.

Upgrades

There are upgrades. I've been drawing them out and planning, see other docs for those. Still brainstorming here.

FRAME gets stronger guns, wider ping, finite shielding, new grenades and eventually instead of taking collision damage, it can ricochet. Mid/late game to add fun and change things up, FRAME can bounce of obstacles and enemies, turning some old areas into new pinball.

KNIGHT gets a shield, an upward boost, a boomerang type thing. Maybe more.

Upgrades aren't done via pause menus, but rather via the garage/shop in the mothership. This needs a picture-in-picture style HUD, and will likely involve multiple cameras overe various scenes rendered out of view in the overworld.

Overworld

Map area ideas: desert (starting area), toxic sea, forest regrowth (difficult flying for drone), fortified canyons, winter/ash, metro/old construction, etc. Look to environmental disasters and climate change for inspiration.

Events

Each overworld area should have some randomized events and moving creatures, little things that emerge from holes and run back in, weird little homebrew bots that scavenge and if killed drop resources. If players decide to drop down and investigate, maybe some new things happen. These are far-future details, but it seems important to the world sim aspect-- and no one wants to play a static/dead game. It needs to feel alive whenever possible.

Weather

The game's all top-down and profile, so no skybox unless its needed for rendering. I do think I'd like to have a weather system (rain, fog, snow, etc) so this means the texture sets need to be built with roughness and some dirt/grime layers exposed for animation. It may be wise to tie the weather systems to the geography and terrain rather than attempting a living system that changes across the whole overeworld.

Goals

Phase I:

1. Continue making notes and getting this thing out on paper. Draw maps, write story beats, sketch it out a bit further.
2. Practice building things with UE4 classes, focus on player controllers.
3. Concept art 1: Get some style frames, photobashes of level designs, Frame
4. Music sketches. Plan a theme, some general tones and ideas for audio during diff't gamestates.
5. Graphic design. Cool off with some logo options and some menu/font choices.

Phase II:

1. Grayboxing and player character swap mechanic builds.
2. Concept art 2: Knight, enemies.
3. Modeling, texturing: Frame, Terrain 1(Overworld start area)
4. Houdini .hip development: procedural enemy drone generator(?)
5. Overworld combat alpha w/ Frame V1

Phase III:

1. Assess state of prototype and needs for next steps. Take a step back.
2. Concept art 3: Mothership, Sidescroller level assets env 1, turrets
3. Modeling, texturing: Knight, Side-scroller environment 1 modules
4. Build and test sidescroller env 1
5. Sidescroller level alpha w/ Knight V1

Notes

The benefit of having almost entirely mechanical assets is limiting the need for rigging, I can take the time I'd spend learning rigging and use that for more Unreal work.

Controls:

Built for controllers—XBOX or PS style analog controllers. Consider mouse and keyboard controls in the future.

Controls should start with controller, and all button functions eventually should be able to be remapped by the user (except the pause button maybe.)

Art style/Color:

Shouldn't be drab and dustbowl, but nor should it be hyperstylized painterly like Blizzard, Dota2 or League of Legends. *Star Wars* (original trilogy) is a great visual reference—colorful, lived-in, old looking future machines. Lattices, greebles, rust-streaked domes, but also lights and electricity everywhere. Deep tones with lots of texture and some saturated highlights and flourishes. Lots of stone, mud, grass, but also lots of scattered semi-campy future tech with some saturated glow. Wires, cables, etc.

Structures and some natural game objects should be destructible, but I don't want to get trapped in the "fabricate all assets to be destroyed," problem. Much of the overworld will be indestructible landmarks. But the player ship has bombs, and those need to have impact. Decals will become necessary as art and mechanics progress. Additionally, enemies being downed should lead to them dropping, bouncing, and exploding. Explosion systems (Houdini?) will be important as they'll be part of the fun and a major way the player gets feedback.

Bibliography

[1] Activision. Battlezone, 1998.

[2] Federal Aviation Administration. Fact sheet – small unmanned aircraft systems (uas) regulations (part 107), 2020.

[3] T. Aiello. New nypd 'digidog' robot raising questions among new yorkers. 2021.

[4] Atari. Battlezone, 1980.

[5] G. Barber and T. Simonite. Some us cities are moving into real-time facial surveillance. 2019.

[6] D. Bavelier. Your brain on video games. https://bit.ly/2ViRxhG, 2012.

[7] A. Blaszczak-Boxe. Drone pilots suffer ptsd just like those in combat. 2015.

[8] M. Busby. Use of 'killer robots' in wars would breach law, say campaigners. 2018.

[9] Capcom. Ghouls'n'ghosts, 1988.

[10] Boston Dynamics. Do you love me? https://bit.ly/3k6cXce, 2020.

[11] Unreal Engine. Unreal engine 4.24 to ship with free quixel megascans, unreal studio features, and more. https://bit.ly/36q6Pn9, 2019.

[12] J. Cameron et. al. The terminator, 1984.

[13] C. Ferguson, S. Rueda, A.Cruz, D. Ferguson, S. Fritz, and S.Smith. Violent video games and aggression: Causal relationship or byproduct of family violence and intrinsic violence motivation? *Criminal Justice and Behavior*, 35:311–332, 2008.

[14] H. Fountain and M. Schmidt. 'bomb robot' takes down dallas gunman, but raises enforcement questions. 2016.

[15] Yacht Club Games. Shovel knight, 2014.

[16] Yacht Club Games. Two million copies of shovel knight sold!! https://bit.ly/3k4mVLo, 2018.

[17] J. Garth. Brusify.io. https://www.brushify.io/, 2020.

[18] A. Griffin. Trump to launch crackdown on violent video games after mass shootings. 2019.

[19] E. Hawkes, L. Blumenschlein, J. Greer, and A. Okamura. A soft robot that navigates its environment through growth. *Science Robotics*, 2017.

[20] [C.] Hocking. Ludonarrative dissonance in bioshock. 2007.

[21] J. Huizinga. *Homo Ludens: A Study of the Play Element in Culture*. Martino Publishing, 1950.

[22] R. Hunicke, M. LeBlanc, and R. Zubek. Mda: A formal approach to game design and game research. 2004.

[23] Irem. R-type, 1987.

[24] Torulf Jernstrom. Let's go whaling: Tricks for monetizing mobile game players with free-to-play. https://bit.ly/36serWo, 2016.

[25] D. Karpukhin. Blockout tools plugin. https://bit.ly/2UvGzVL, 2019.

[26] Konami. Castlevania, 1986.

[27] B. Levinson and V. Curtin et. al. Toys, 1992.

[28] Rare Ltd. Wizards and warriors, 1987.

[29] T. Maughan. The dystopian lake filled by the world's tech lust. 2015.

[30] K. Miles. Artificial intelligence may doom the human race within a century, oxford professor says. 2014.

[31] Nintendo. The legend of zelda, 1986.

[32] Nintendo. Kid icarus, 1987.

[33] Rockstar North. Grand theft auto v, 2015 (Windows Version).

[34] Nintendo of America Inc. Nintendo power awards '88. *Nintendo Power*, 1989.

[35] The Bureau of Investigative Journalism. Drone warfare. https://bit.ly/2UDylLr, 2010-2020.

[36] Rivers of Steel. Carrie blast furnaces national historic landmark. https://bit.ly/36FEsBL, 2021.

[37] M. Pavlovich. Houdini auto game res. https://bit.ly/3e2Z4aR, 2020.

[38] [A.] Pratt. Cluedo/clue, 1949.

[39] Associated Press. Former nfl player kills 6 people, then himself, in south carolina. 2021.

[40] CD Projekt Red. The witcher 3: Wild hunt, 2015 (Windows Version).

[41] [M.] Robbins. Uno, 1971.

[42] SEGA. Herzog zwei. https://segaages.sega.com/project/herzog-zwei/.

[43] Bethesda Game Studios. The elder scrolls iii: Morrowind, 2002.

[44] Bethesda Game Studios. The elder scrolls v: Skyrim, 2011 (Windows Version).

[45] Rockstar Studios. Red dead redemption 2, 2019 (Windows Version).

[46] Technosoft. Herzog zwei, 1990.

[47] K. Telinbas and E. Zimmerman.

[48] Campaign to Stop Killer Robots. Who wants to ban fully autonomous weapons? https://www.stopkillerrobots.org/, 2021.

[49] J. Uitti. Nike nuclear missile site s-13/14. https://bit.ly/3ww3FJl.

[50] J. Vincent. The nypd is sending its controversial robot dog back to the pound. 2021.

[51] A. Vintsevych. Witch hunt, 2018.

[52] D. Zendle, P. Cairns, and D. Kudenko. No priming in video games. *Computers in Human Behavior*, 78:113–125, 2018.