8-2017

Kingdom Rush Tribute: Porting a 2D Tower Defense Game to a 3D World using Unreal Engine 4

Christian Stith
Clemson University, cstith@g.clemson.edu

KINGDOM RUSH TRIBUTE: PORTING A 2D TOWER DEFENSE GAME TO A 3D WORLD USING UNREAL ENGINE 4

A Thesis
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
Christian Stith
August 2017

Accepted by:
Dr. Brian A. Malloy, Committee Chair
David Donar
Dr. Eric Patterson

Abstract

Video game remakes are a popular modern phenomenon in which an existing game is remade to target a different platform, a higher resolution, or simply a different audience. Seldom, however, are video games remade into a different genre altogether. In this paper, we describe the process of remaking Kingdom Rush, a popular top-down 2D tower defense game, into a 3D third-person shooter in Unreal Engine 4. In addition to recreating all visual components of the game in 3D, this process revealed several challenges that arose related to transferring the gameplay from a 2D screen to a 3D world. We describe these challenges and their solutions and describe the results of our work: a playable, single-level prototype that mimics the original game in aesthetics and functionality while also incorporating a controllable hero character.

Table of Contents

Table of Contents (Continued)

Page

List of Figures

List of Figures (Continued)

Page

Chapter 1

Introduction and Motivation

For the king!

The Reinforcements

The video game industry has grown from a curiosity to a full fledged entertainment industry to the extent that in 2016, video game revenues surpassed both music and movie industry revenues [5]. The gaming industry has grown so quickly that by the end of 2017, global video game revenues are projected to exceed 91 billion dollars [2]. During much of its history, video game play has been dominated by hard core gamers who were willing to spend hours crafting a large empire or honing their hand-eye coordination for high speed first person shooters [3]. However, recently a new class of video game, the *casual game*, has found immense appeal among a vast audience that would not be characterized as hard core gamers. For example, games such as *Bejeweled*, *Plants and Zombies*, *Angry Birds*, *Slotomania*, and more recently, *Pokemon Go*, have become attractive to casual gamers. One genre of casual games that has appealed to a wide audience is the *Tower Defense* genre, where the goal of the Tower Defense game is to defend the player's territories or possessions by obstructing or terminating the enemy attackers, usually achieved by placing defensive structures along their path of attack.

A feature common to most Tower Defense video games is that most of these games use side scrolling, isometric, or top-down perspective graphics, and are characterized by 2D or 2.5D game play [7]. However, in October of 2010, *Dungeon Defenders* was released as one of the first tower defense games to incorporate the third person perspective to the tower defense genre. *Dungeon Defenders* achieved sales of over 250,000 copies in the first two weeks of release [1] and over 600,000 copies by the end of 2011 [6], illustrating the tremendous potential for attracting casual gamers to the genre when ported to a 3D environment. *Dungeon Defenders* was followed in 2011 by the popular tower defense game *Orcs Must*

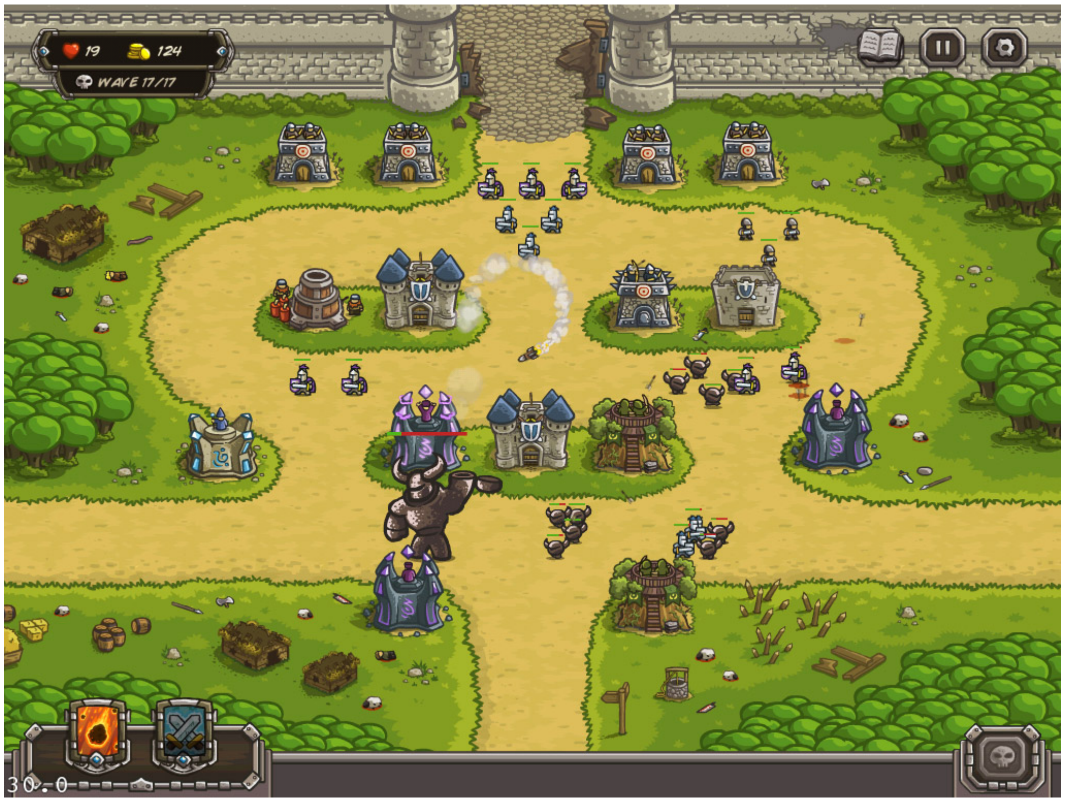


Figure 1.1: The original Kingdom Rush game.

Die!, which shifted the tower defense game to a first person perspective. Nevertheless, the majority of tower defense games remain 2D. One such game is Kingdom Rush, a very popular tower defense game published by Ironhide Studios. A screen capture of a typical level in the original Kingdom Rush genre is illustrated in Figure 1.1, showing fourteen towers and a curved path traveling from the entrance at the bottom to the exit at the top of the figure.

In this thesis we describe our adaptation of a popular 2D tower defense game, Kingdom Rush, to a 3D platform using the Unreal Game Engine. We call the resulting prototype Kingdom Rush: Tribute. In Tribute, the player controls an in-game hero who travels throughout the map building towers and fighting enemies. We describe the architectural and artistic design processes of this adaptation, and discuss some new challenges that appear in the development of a 3D version that would not have appeared in the 2D version. In the next chapter we define terminology and describe tools and applications that we use in this thesis. In Chapter 3 we provide an overview of our work and in Chapter 4 we provide some implementation details in using the Unreal Game Engine. In Chapter 5 we describe our design and development of Tribute assets, and in Chapter 6 we describe some of the challenges in translating a game from a 2D environment to a 3D world, and the corresponding solutions. In Chapter 7 we describe some anecdotal feedback that we received from playtesting and the changes that were informed by this feedback. Finally, in Chapter 8 we provide concluding remarks and describe our future work.

Chapter 2

Background

Prepare for glory!

Sir Gerald Lightseeker

In this chapter, we provide background information and describe terminology that we use in this thesis. In the next section we describe the video game framework that we use, the Unreal Engine, referred to as UE4. In Section 2.2 we describe camera placement in various game genres, and in Section 2.3 we describe the various genres.

2.1 Unreal Engine

Unreal Engine is a professional game development engine produced by Epic Games. The engine is best known for its deferred renderer and its Blueprints visual scripting system. A sample Blueprint is provided in Figure 2.1, which illustrates the visual nature of Blueprint. Beginning in 2014, Epic Games released the full version of its engine as freeware, which catapulted the engine's popularity and versatility with indie and student developers. Unlike many game engines, UE4 is unique in that it can be used extensively without comprehensive use of computer programming. While the engine does have full support for C++ coding, UE4 users can also design and implement games using the Blueprints visual scripting system. While the logical functionality of this system is not as powerful as pure coding, using Blueprints nevertheless permits artists and designers to quickly develop and preview visual changes to in-game assets.

For this project, we make use of Unreal Engine version 4.13. The choice of Unreal was driven primarily by the option to use the Blueprints system. The version was chosen primarily for its support of the Alembic file format, which allows for complex vertex animations such as precomputed rigid body simulations.

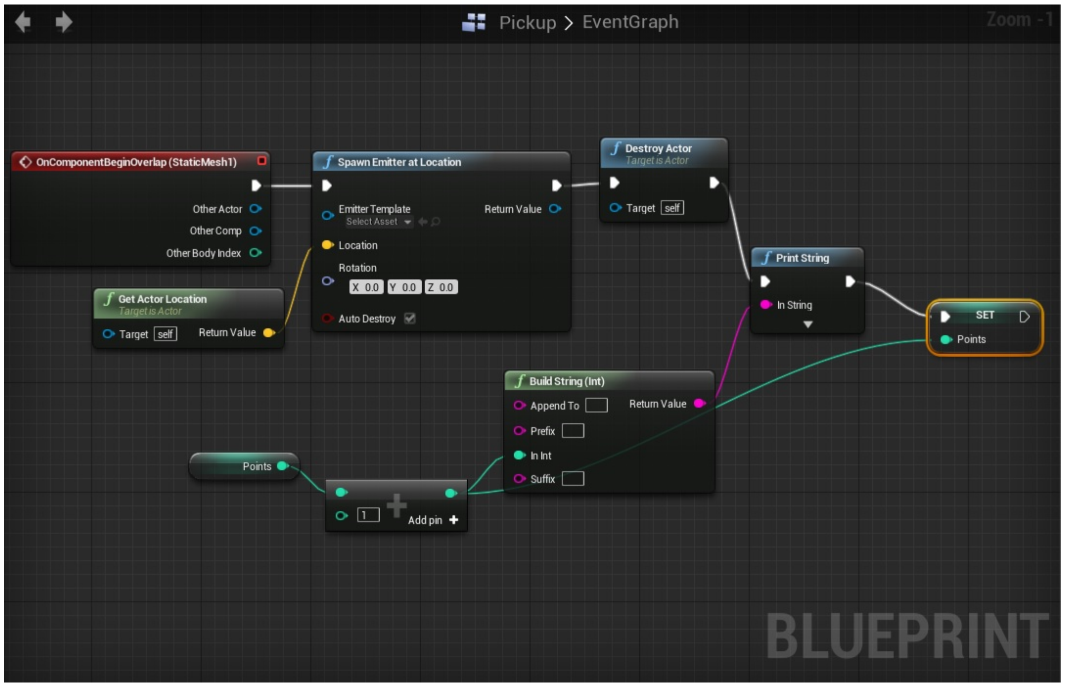


Figure 2.1: A UE4 Blueprint.

2.2 Camera Placement

While video games are most commonly differentiated based on their genre, another differentiating factor is the placement of the render camera. We describe several of the relevant setups below.

2.2.1 Fixed Overhead

In the *fixed overhead* approach, video games make the entire field of play visible at all times, and make use of a stationary, downward facing camera. In these systems, the camera is placed high above the playspace and provides the player with a comprehensive birds-eye view of the game. Aside from brief cinematic montages, the camera usually does not move during gameplay. This style of camera is used in the original Kingdom Rush game, as can be seen in Figure 2.2.

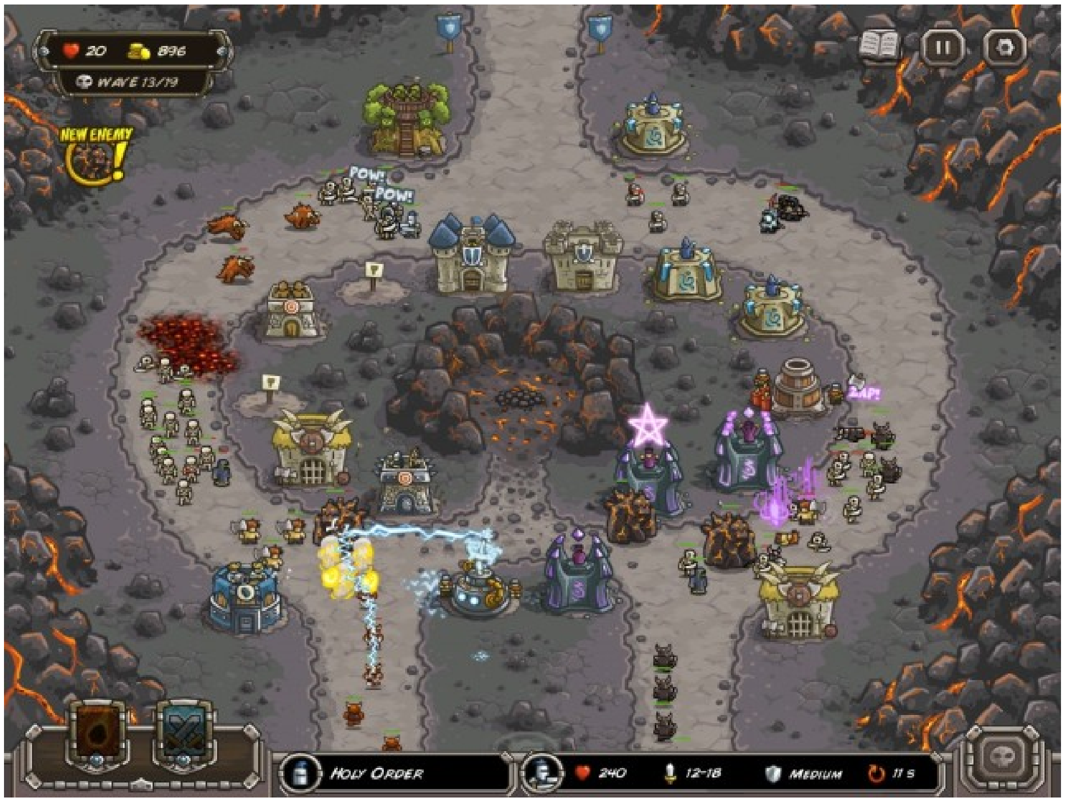


Figure 2.2: An overhead camera view from the game Kingdom Rush.

2.2.2 Third Person

The *third person* approach to camera placement is commonly implemented in action and shooter games, third person camera placement involves a dynamic camera that follows the player's controllable character around the playspace from a set distance. This camera position allows the player to see their entire character and the general area around it in all directions. Such cameras are often include spring arm functionality that allows the camera to automatically react to geometry that blocks the player character from the camera's view by moving into a better position. An example of this camera view is shown in Figure 2.3.

2.2.3 First Person

Similar to the third person camera setup, *first person* cameras follow the player character around the map. Instead of following the player character via a spring arm, however, a first

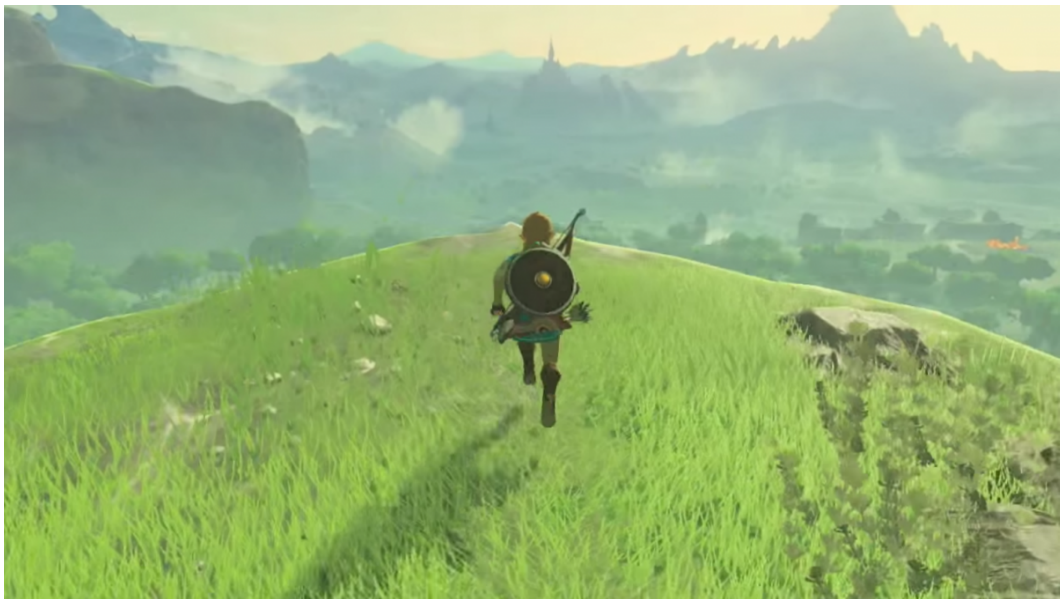


Figure 2.3: A third-person view from the upcoming game Zelda: Breath of the Wild.

person camera is mounted on the character's neck, providing the player with a view that simulates what the character would see in the game. An example of this camera view can be seen in Figure 2.4. Using a first person camera often necessitates developing a different character model, rig, and animation system in order to present a convincing and appealing point-of-view frame. Many action games offer the option to toggle between third-person and first-person camera setups.

2.3 Video Game Genres

Like any form of media, video games are often classified into different genres, based mainly on the style of gameplay. These main genres often encompass other more specialized sub-genres, and video games often blur the lines between genres by mixing different aspects of gameplay from different genres. While the original Kingdom Rush game has a clearly defined genre, Kingdom Rush: Tribute is best described as a hybrid of several video game genres, each of which we will describe here.

Figure 2.4: A first-person view from the classic shooter game Doom.

2.3.1 Real-Time Strategy

Real-Time Strategy, or RTS, is a subgenre of strategy video games in which the gameplay does not occur in alternating turns. Instead, actions occur on the part of the players concurrently, and each player must decide how to best make use of limited resources in their attempt to defeat their opponents.

2.3.2 Tower Defense

Tower defense, or TD, is a real-time strategy video game genre in which the player tries to stop a group of enemies from reaching a target by building functioning towers that impede and destroy them. Players typically gain resources by destroying enemies, allowing them to upgrade existing towers and build more towers to combat increasingly more frequent and powerful enemies. In most such games, enemies occur in a predetermined pattern for each level, and travel down fixed paths toward the target. Popular games in this genre include Bloons, Frozen Islands, Desktop Tower Defense, and Flash Element Tower Defense.

2.3.3 Shooter

Shooter games are a subgenre of action games in which the player's character is armed with a ranged weapon. Shooter games can be third or first person, and usually feature fast-paced gameplay that requires the player to battle a set of aggressive enemies. War games such as the Quake, Doom, Half-Life, Call of Duty, and Halo franchises fall into this category.

2.3.4 Kingdom Rush: Tribute

Tribute blends elements from each of the genres described above. Despite the shift in gameplay style, the majority of the tower defense features remain in the game, and building strong towers is still essential to defeat the attackers. However, the camera has been modified from an overhead view, featuring first and third person cameras: both hallmarks of action games. The addition of a controllable character with ranged weapon and melee combat abilities also place the game in the tradition of action/shooter games. We believe that Tribute is best defined as a crossover game that combines the tower defense and action genres.

Chapter 3

Project Overview

Have at thee!

The Soldiers

Kingdom Rush is a series of tower defense games developed by Ironhide Studios. Originally released as a Flash game on the website ArmorGames.com, the game quickly became the highest-rated game in the history of the website. Following the success of the original Kingdom Rush game, Ironhide released a sequel, Kingdom Rush: Frontiers, and a prequel, Kingdom Rush: Origins. Both games were released for iPhone and Android, and Kingdom Rush and Frontiers were later released on Steam. [8]

While Kingdom Rush and both of its sequels are highly rated, enjoyable games, experienced players will be familiar with several frustrating aspects of gameplay. In particular, the inability to directly control the hero character results in this character's powerful melee abilities often being wasted against weaker enemies. We can recount many instances in which the hero character ignored a powerful boss in favor of a much weaker goblin, allowing the boss to pass by and resulting in a much lower final score. It is these instances which inspired us to create a new version of the game - one that allowed the player complete control over the hero while still maintaining the original Kingdom Rush gameplay.

Kingdom Rush: Tribute is a three-dimensional first-person prototype remake of the original Kingdom Rush game. Tribute is built in Unreal Engine 4.13 and uses assets developed in Maya 2016, Houdini 15, and Photoshop CC. In Tribute, rather than playing from a birds-eye-view position, the player directly controls a Hero character who can engage enemies in battle, fire ranged weapons, and purchase, upgrade, and sell towers. The player controls his character through the use of standard PC action keyboard and mouse controls rather than using the point and click interaction method of the KR games.

Tribute takes place in the Citadel level, and implements all of the towers and most of the enemies that occur in that level in the original game. While the Citadel is currently the only developed level, the game design architecture places a strong emphasis on scalability, and adding new levels and enemies is simply a matter of generating the assets themselves. We describe some of the necessary improvements we hope to achieve in Chapter 8.

Chapter 4

Project Implementation

Want some? Get some!

The Dwarves

In this chapter we describe the implementation of our prototype of Kingdom Rush: Tribute. In the next section we describe our effort to maintain fidelity to the original Kingdom Rush series while porting the 2D series to a 3D platform. In Section 4.2 we review the design and implementation of the characters that we have incorporated into the prototype. In Section 4.3 we describe the towers that we have incorporated into the prototype. Finally, in Section 4.4 we present our architecture and class hierarchy.

4.1 Fidelity to the Original Kingdom Rush Series

In order to create a convincing Kingdom Rush experience, it is necessary to stay as true to the original game as possible. This involves matching both the visual aesthetic of the game as well as the statistical and functional aspects of the gameplay. To address the visual aesthetic aspect, we take a derivative approach to the development of the assets, treating the original game sprites as source material and concept art. To address the statistical element of gameplay, we draw from gameplay knowledge, in-game information, and game wikis to ensure that the characters, towers, and interaction are as close to the original game as possible. The following sections describe this process in further detail.

4.2 Characters in Kingdom Rush: Tribute

Kingdom Rush features a wide array of characters, the bulk of which are either defenders or enemies. The original game implements seventy different types of enemies, eleven of which are bosses. The other fifty-nine are regular enemies that spawn in large numbers throughout levels and are equipped with a variety of mechanics, including walking, running, flying, idling, melee attacks, area attacks, ranged attacks, healing, dodging, and dying. To implement these mechanics, KR makes use of *sprite sheets*, allowing each character to be represented by a sequence of still images corresponding to the current state of the character. A new enemy can be created by selecting the mechanics needed for the character and creating the appropriate sprite sheets. As 2D sprites are not an effective way of rendering characters in 3D, it is necessary to develop a 3D character system to transfer the enemy system into the 3D world. To achieve this, we modeled, textured, rigged, and animated 3D versions of each character using Maya 2016 and Photoshop CC. This process is described in more detail in Chapter 5.

Once each character's artistic assets were created, it was necessary to provide the character with a controller system that would drive its behavior as it interacted with the game world. In order to accommodate the high number of enemy types, the system needed to be highly scalable and allow for interchangeable elements. We implemented this character controller system using an object-oriented hierarchical class structure that is based on the UE4 Pawn class, which provides basic character movement functionality. We created a single base class, Rusher, which inherits directly from the Pawn class. All Kingdom Rush: Tribute characters are descendants of this Rusher class, which includes universal functionality such as goal seeking, ranged projectile firing, damage taking, health, a skeletal mesh, clothing, and weapons. Immediate children of the Rusher class include the Defender and Attacker classes. All Tribute soldiers and the player's hero character are descendants of the Defender class, while all attacking enemy characters are descendants of the Attacker class.

4.2.1 Rusher Class

The Rusher class implements the functionality common to all KR characters, including navigation, animation, health, combat, and ranged weaponry. These functions can be customized for individual characters by overriding the default values of the Rusher variables. For example, Defender classes override the ShouldRespawn variable from false to true, causing those classes' Die function to trigger a delayed respawn event. Any additional functionality implemented by a character occurs in that character's class. Examples of additional functionality include special attacks such as the Paladin's Holy Strike or the Juggernaut's Golem Bomb Launch. Rather than supply an exhaustive list of the class variables and functions common to all Rushers, we describe the primary class functionality in the following list:

- Skeletal Mesh: the rigged model of the Rusher. To this skeletal mesh we can attach a variety of components such as a sword, a bow, a helmet, a vest, and other accessories as each character requires. The skeletal mesh is controlled by an animation blueprint.
- Animation Blueprint: controls the skeletal mesh's animation. By polling the current state of the Rusher's variables, the animation blueprint determines what animation should be playing, and transitions the skeletal mesh to that animation appropriately. The animation blueprint also triggers certain events, such as applying damage to a target at the appropriate point in the attack animation cycle. Tribute has one unique animation blueprint for each type of rig.
- Update: this function is called every frame, and updates the Rusher's health bar, checks to see if the Rusher should be dying, orients the Rusher to face its opponent when appropriate, and performs other housekeeping functions.
- Give Damage: this function sends the appropriate amount and type of damage to the Rusher's opponent. This function is usually called by the animation blueprint when the Rusher's melee weapon is swung.
- Take Damage: this function processes incoming damage. This function does a switch on the incoming damage type, decreases the incoming damage value by the Rusher's magical or physical armor values as appropriate, and subtracts the resulting value from the Rusher's health. If the health reaches 0 or less, the Die function is called.
- Die: this function kills the Rusher, usually as a result of the Rusher's health reaching

zero. In most cases, this results in the Rusher's skeletal mesh playing its death animation, followed by the destruction of the actor. If the Rusher is killed by explosive damage, it is destroyed instantly, and a blood spatter decal is spawned at the Rusher's location on the map.

- Check Range: if the Rusher is equipped with a ranged weapon and is currently not engaged in melee combat, this function polls the Rusher's range sphere to see if any opposing Rushers are within range. If so, one of these Rushers will be chosen as the RangedTarget, and the Rusher will begin firing its ranged weapon.

- Fire Ranged Weapon: this function calculates the appropriate trajectory needed to hit the RangedTarget opponent and fires a ranged projectile at that trajectory. It is usually called by the animation blueprint at the appropriate point in the skeletal mesh's firing animation sequence.

- Set Opponent: this function sets the Rusher's current melee opponent, engaging them in combat. It is usually called by a Defender seeking to initiate combat.

- Take Wrath: This function initiates the Rusher's Wrath of the Forest struggle animation. It is usually called by an Archer Tower of level 4 when the Wrath of the Forest special attack is fired.

4.2.2 Enemies

Attacker

In addition to the functionality of the Rusher class, the Attacker class features path following ability. This functionality is achieved by adding a child Goal actor and overriding the Rusher Update function to ensure that the Attacker is always following the Goal. The Attackers implemented in Tribute are described in the following paragraphs.

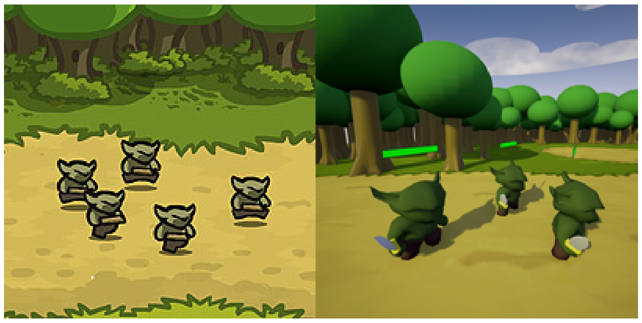


Figure 4.1: KR Goblins, left, and Tribute Goblins.

Goblin

The Goblin class includes the Goblin skeletal mesh, rig, and animation blueprint. Goblins have an HP of 20, deal physical damage in the range of 1-4, have no physical or magical armor, run at medium speed, earn 3 gold when killed, and cost 1 life if they reach the goal point. They have no special abilities and do not respawn. In-game screenshots of Goblins can be seen in Figure 4.1.

Bandit

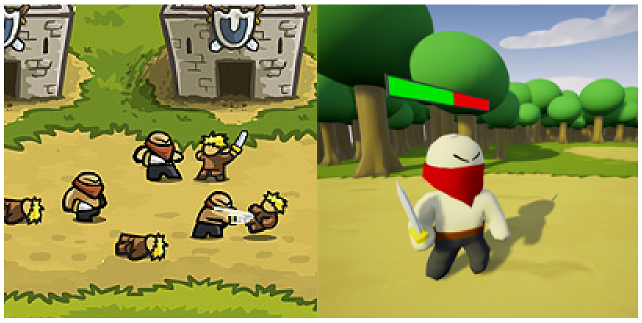


Figure 4.2: KR Bandits, left, and a Tribute Bandit.

The Bandit class includes the Biped skeletal mesh and rig and the BipedAttacker animation blueprint. Bandits have an HP of 70, deal physical damage in the range of 20-30, have no physical or magical armor, run at medium speed, earn 8 gold when killed, and cost 1 life if they reach the goal point. They have no special abilities and do not respawn. In-game screenshots of Bandits can be seen in Figure 4.2.

Brigand

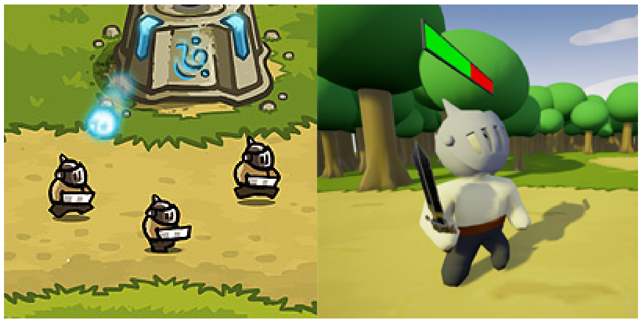


Figure 4.3: KR Brigands, left, and a Tribute Brigand.

The Brigand class includes the Biped skeletal mesh and rig and the BipedAttacker animation blueprint. Brigands have an HP of 160, deal physical damage in the range of 6-10, have medium physical armor and no magical armor, run at medium speed, earn 15 gold when killed, and cost 1 life if they reach the goal point. They have no special abilities and do not respawn. In-game screenshots of Brigands can be seen in Figure 4.3.

Shadow Archer

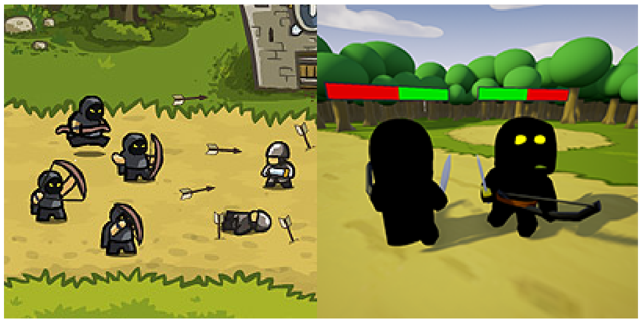


Figure 4.4: KR Shadow Archers, left, and Tribute Shadow Archers.

The Shadow Archer class includes the Biped skeletal mesh and rig and the BipedAttacker animation blueprint. Shadow Archers have an HP of 180, deal physical damage in the range of 10-20, have no physical armor and medium magical armor, run at medium speed, earn 16 gold when killed, and cost 1 life if they reach the goal point. They have a ranged arrow attack, and will stop marching to destroy any defenders from long range before continuing. They do not respawn. In-game screenshots of Shadow Archers can be seen in Figure 4.4.

Gargoyle

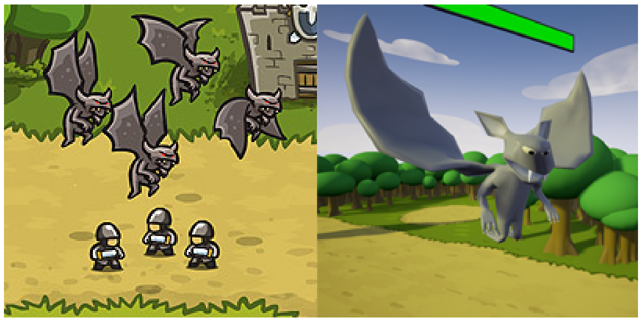


Figure 4.5: KR Gargoyles, left, and a Tribute Gargoyle.

The Gargoyle class includes the Winged skeletal mesh, rig, and animation blueprint. Gargoyles have an HP of 90, have no physical armor and no magical armor, fly at medium speed, earn 12 gold when killed, and cost 1 life if they reach the goal point. Gargoyles are flying creatures, so they cannot be engaged by barracks troops and do not deal any damage. They have no special abilities and do not respawn. In-game screenshots of Gargoyles can be seen in Figure 4.5.

Juggernaut

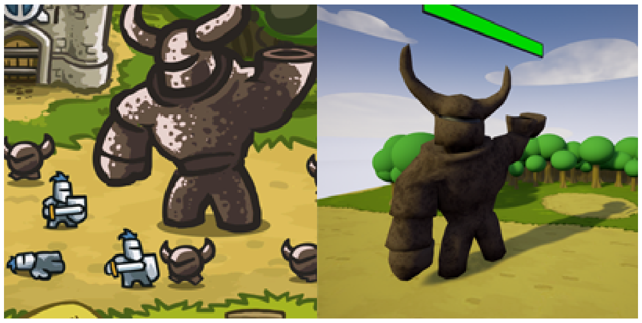


Figure 4.6: The KR Juggernaut, left, and the Tribute Juggernaut.

The Juggernaut class includes the Juggernaut skeletal mesh, rig, and animation blueprint.

The Juggernaut has an HP of 10,000, deals physical damage in the range of 150-250, has no physical armor and no magical armor, walks at slow speed, does not earn gold when killed, and costs 20 lives if it reaches the goal point. The Juggernaut is a boss enemy who appears only on the final wave of The Citadel. Defeating this enemy automatically wins the level for the player, while letting him reach the goal point automatically loses the level. The Juggernaut has a unique ranged weapon that can hit any path location, dealing area damage and spawning several Golem Heads upon impact. In-game screenshots of The Juggernaut can be seen in Figure 4.6.

Golem Head

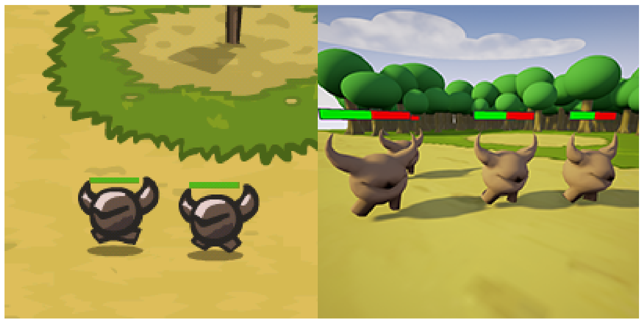


Figure 4.7: KR Golem Heads, left, and Tribute Golem Heads.

The Golem Head class includes the Mini skeletal mesh, rig, and animation blueprint. Golem Heads have an HP of 125, give physical damage in the range of 10-20, have no physical or magical armor, run at slow speed, earn 10 gold when killed, and cost 1 life if they reach the goal point. They have no special abilities and do not respawn. Golem Heads appear only when spawned by The Juggernaut, and never enter the map from on their own like other enemies. In-game screenshots of Golem Heads can be seen in Figure 4.7.

4.2.3 Defenders

In addition to the functionality of the Rusher class, the Defender class features health regeneration, which occurs any time the Defender is idling. Defenders also have an Army to which they can belong, which allows groups of three defenders to move as a single unit. Further Defender logic is described in section 4.3.3. The defenders implemented in Tribute are described in the following paragraphs:

Militia

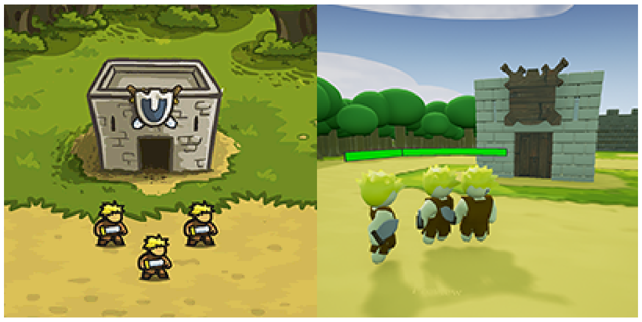


Figure 4.8: KR Militia, left, and Tribute Militia.

The most basic of KR defenders, Militia spawn from Militia Barracks in groups of 3. Milita deal damage in the range of 1-3, have an HP of 50, a respawn time of 10 seconds, and have no physical or magical armor. In addition to inhabiting the lowest level of barracks tower, Militia are also spawned in pairs when the Hero calls for reinforcements. Militia have no special abilities and no ranged attack. In-game screenshots of Militia can be seen in Figure 4.8.

Footman

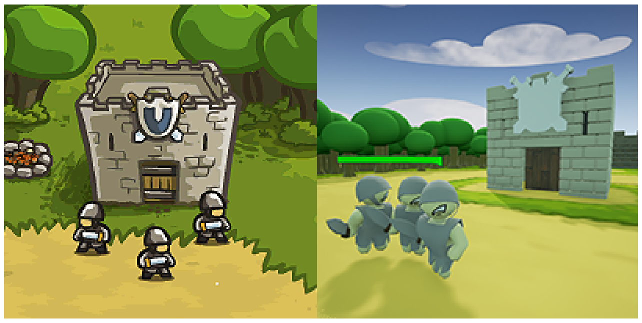


Figure 4.9: KR Footmen, left, and Tribute Footmen.

The second level of defender, Footmen spawn from Footmen barracks in groups of 3. Footmen deal damage in the range of 3-4, have an HP of 100, a respawn time of 10 seconds, and have low physical armor and no magical armor. Footmen have no special abilities and no ranged attack. In-game screenshots of Footmen can be seen in Figure 4.9.

Knight

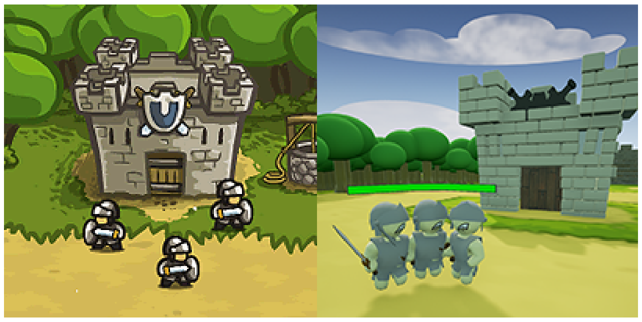


Figure 4.10: KR Knights, left, and Tribute Knights.

The third level of defender, Knights spawn from Knights barracks in groups of 3. Footmen deal damage in the range of 6-10, have an HP of 150, a respawn time of 10 seconds, and have low physical armor and no magical armor. Knights have no special abilities and no ranged attack. In-game screenshots of Knights can be seen in Figure 4.10.

Paladin

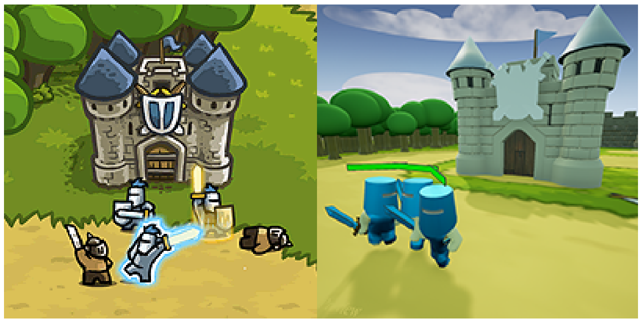


Figure 4.11: KR Paladins, left, and Tribute Paladins.

The first of the two highest levels of defender, Paladins spawn from Holy Orders in groups of 3. Paladins deal damage in the range of 12-18, have an HP of 200, a respawn time of 14 seconds, and have medium physical armor and no magical armor. Paladins features the Holy Strike special attack, which is described in section 4.3.3. Paladins do not have a ranged weapon. In-game screenshots of Paladins can be seen in Figure 4.11.

Hero

The Hero class includes the Biped skeletal mesh and rig and the Hero animation blueprint. Unlike other Rusher implementations, the Hero's controller does not implement any automatic functionality. Instead, the Hero's motion is controlled by the player, who inputs directional commands through the WASD keys, jumping commands by pressing the spacebar, and weapon usage by clicking the right and left mouse buttons. An in-game screenshot of the Hero character can be seen in Figure 4.12. In addition to providing character move-

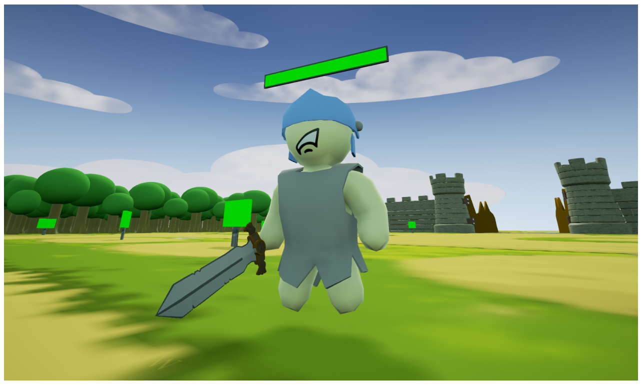
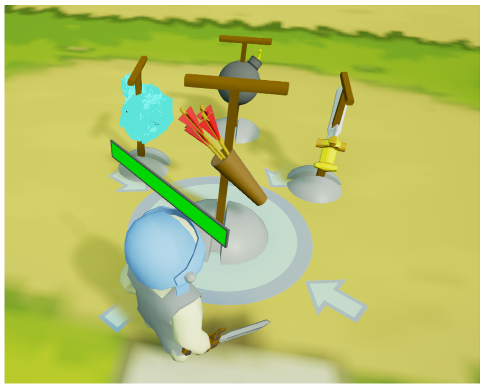


Figure 4.12: The Tribute Hero character.

ment and battle functionality, the Hero character also provides the player with the means to build, upgrade, and sell towers. We accomplish this through a new method of interaction detection that is not present in the original game. In addition to the basic rusher collision capsule, the Hero class also includes a small collision sphere directly ahead of the skeletal mesh that identifies what object the player is currently interacting with. If the collision sphere overlaps an interactable object, a KRInteraction decal object appears a that object's position, highlighting that object for the player, as shown in figure 4.13 .


(a) The player can pull this lever to build a Tower.

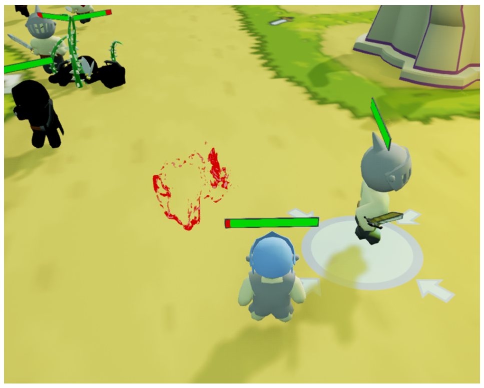

(b) The player can attack this Brigand.

Figure 4.13: The interaction decal in Kingdom Rush: Tribute.

If the player wishes to interact with this object they do so by clicking the left mouse button. If the chosen object is a building lever, the lever is pulled and the appropriate action occurs. If the chosen object is an Attacker, the Hero engages the Attacker in melee combat. The player can also switch between third-person and first-person view in order to fire ranged projectiles instead of using a melee weapon. The Tribute Hero is loosely based on Gerald Lightseeker, a hero from the original KR game. Before leveling up his stats, Gerald Lightseeker has 400 HP, deals 11-18 damage, has low physical armor and no magical armor, and has no special abilites. Once leveled up, he unlocks the special abilites Courage and Shield of Retribution. As we have not yet implemented a player progress system, we have chosen not to implement these special attacks.

4.3 Towers

Kingdom Rush features four main tower types: Archer Towers, Mage Towers, Barracks, and Dwarven Bombards. Each of these towers features a different attack type, and each can be upgraded twice before a final upgrade into one of two Advanced Towers. The visual aspects of the towers in Kingdom Rush: Tribute are based heavily off of the original game, with creative liberties taken as needed or desired. In particular, details such as idle animations and particle emitters are added where source detail is found to be lacking. To represent KR towers in Tribute, meshes were modeled in Maya 2016 and textured in Photoshop CC. The Mage Towers and Archer Towers were built as static meshes, while the Barracks and Dwarven Bombards were given skeletal components to accommodate the animations associated with their respective action sequences. Each tower is manned by a set of characters represented by animated skeletal meshes. To replicate the sprite sheet animation of the original towers, each tower is given an action sequence. These sequences involved a range of effects, from skeletal animation to particle emitters. In addition to the basic attack featured by the basic towers, the Advanced towers each feature two or three Special Attacks. These specialized attacks have a longer cooldown time, but greatly enhance the ability of the towers to impede and destroy enemies. A crucial aspect of Kingdom Rush strategy is deciding which Advanced towers will be most effective at defeating a particular set of enemies. The four basic towers, their visual effects, their attack types, their special attacks, and their relative strengths and weaknesses are outlined in the sections below.

4.3.1 Archer Tower

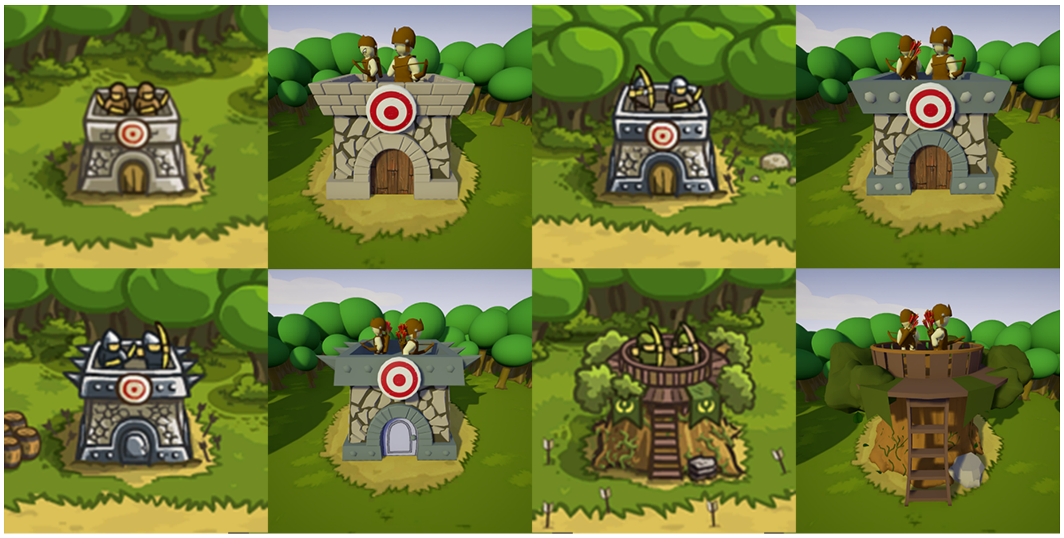


Figure 4.14: Kingdom Rush Archer Towers and their corresponding Tribute Archer Towers.

Overview

The archer tower has a base cost of 70 gold, and features a fast ranged attack that deals physical damage and targets both ground and flying enemies. The projectiles launched by the attack do not deal area damage and are not homing. The damage dealt by archer tower arrows is fairly low, but the rate of fire of this tower is the highest in the game. In-game screenshots of archer towers can be seen in Figure 4.14.

Action Sequence

Each iteration of the archer tower features two biped archers which are responsible for the tower's attack function. When an Attacker comes within range of the tower, the first archer initiates his firing sequence. At the appropriate point in the firing animation cycle, an Arrow projectile is spawned at the front of the bow with an initial horizontal velocity of 20 meters per second. The arrow is aimed by the predictive aiming function described in section 6.1, and will miss its target if the target Attacker is banking significantly. At the halfway point in the first archer's firing cycle, the second archer initiates his firing sequence, resulting in an alternating series of fired arrows. If an arrow collides with an Attacker, it is destroyed, and passes the appropriate amount of physical damage to the Attacker's TakeDamage function.

VFX

The basic levels of the archer tower feature no visual effects. The Ranger's Hideout advanced tower features one leaf emitter to enhance the visual appearance of the tower, an aspect which is not present in the original game.

Special Attacks

The Ranger's Hideout tower features Wrath of the Forest, a timer-based special attack that briefly traps a set of enemies in place, dealing them up to 45 damage over 3 seconds. This attack is implemented in KR with a briar skeletal mesh. When the special attack is triggered, a briar will spawn directly beneath up to five enemies within range of the tower, along with a particle leaf burst for effect. Each chosen Attacker enters into a struggling animation that matches the briar's attack animation, and takes physical damage throughout the attack. Once the attack ends, the briar retreats into the ground, and the Attacker returns to his previous behavior.

4.3.2 Mage Tower

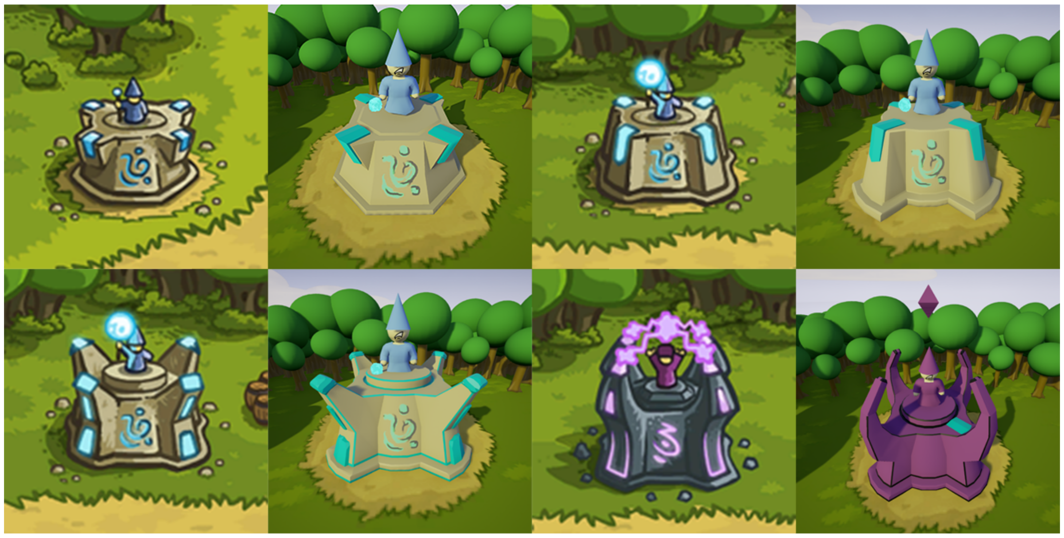


Figure 4.15: Kingdom Rush Mage Towers and their corresponding Tribute Mage Towers.

Overview

The Mage Tower has a base cost of 100 gold, and features a slow ranged attack that does magical damage and targets both ground and flying enemies. Projectiles fired by this tower do not deal area damage but do home to their target. The base damage for mage towers is relatively high. In-game screenshots of Mage Towers can be seen in Figure 4.15.

Action Sequence

The Mage Tower is manned by a single wizard at all levels. When an Attacker comes within range of the tower, the wizard begins the spellcasting animation sequence, spawning a MageBolt from the tip of his wand. The MageBolt is a homing projectile, and cannot miss its target. If the target Attacker is destroyed before the bolt reaches him, the bolt will crash harmlessly into the ground. Otherwise, the bolt spawns an explosion upon colliding with the Attacker, and passes appropriate magic damage to the Attacker's TakeDamage function.

VFX

The mage tower meshes feature magical gems that pulse in intensity prior to the wizard's attack, an effect achieved with a dynamic scalar parameter piped to the gem shader emission attribute. Both the wizard's wand tip and the MageBolt feature a noisy displacement shader that is colored via cross panning noise piped into a stepped blue ramp to achieve a magical toon look. The MageBolt features the same shader, as well as a tapering particle trail system to mimic the comet trail look of the original game sprites. The impact explosion features a toon sprite sheet created in Houdini 15.

Special Attacks

The Arcane Wizard advanced tower features a different basic attack that involves a magic beam in a straight line from the tower tip to the Attacker. This effect is preceded by a set of electric purple sprites crawling up the tower spires. These sprites were procedurally generated using Houdini 15. Once the sprites reach the top of their respective spires, four beam particles are emitted that meet in the center of the spires, and a fifth beam spawns between the center point and the target Attacker. The beam particles are given a noisy displacement to mimic the electric look of the original game sprites.

4.3.3 Barracks

Overview

The Barracks tower has a base cost of 70 gold, and unlike the other three towers, features no ranged attack. Instead, a barracks spawns three soldiers which center around an adjustable

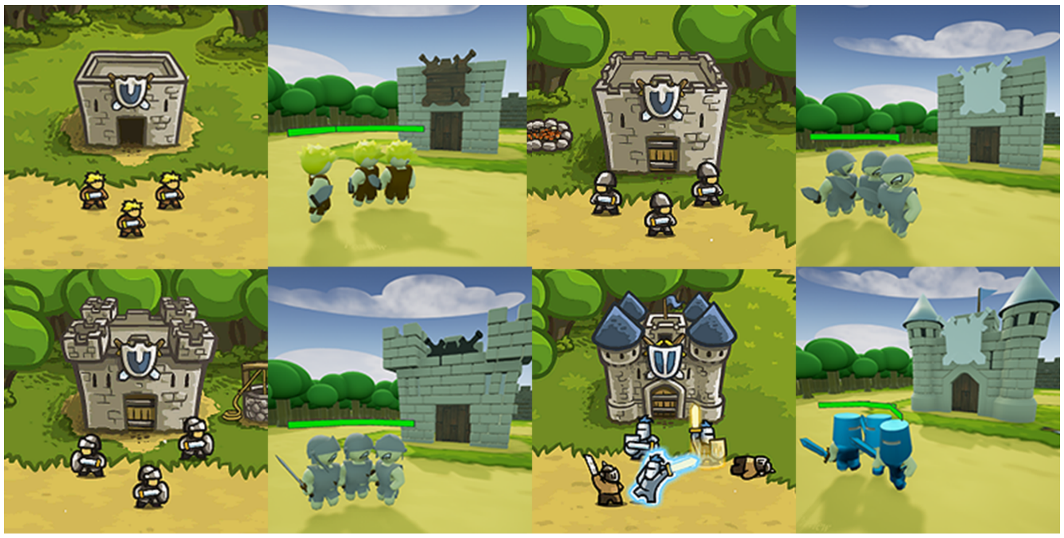


Figure 4.16: Kingdom Rush Barracks and their corresponding Tribute Barracks.

rally point and engage enemies that try to pass them in combat. When killed, each soldier will respawn after a brief recharge period (10-14 seconds). In their base form, barracks soldiers cannot target or engage flying enemies, have relatively low health, and deal low amounts of physical damage to attackers. In-game screenshots of Barracks can be seen in Figure 4.16.

Action Sequence

The animation sequence of the barracks is straightforward, since it has no attack function, and presents a trigger that opens the door to release any new soldiers that spawn within. Once the barracks is empty, the door closes. The rest of the barracks functionality is performed by the three soldiers who spawn inside it. The barracks soldiers are instances of the Defender class and are spawned by an Army, which assigns them both a spawn point and a rally point. Once spawned, Defenders return to the rally point unless an Attacker is found within range of the Army. When an Attacker is found within range, all three Defenders immediately navigate to it and begin attacking it. The Attacker will choose one of the defenders and begin to fight back. When another Attacker enters within range, a Defender who is not mutually engaged (being attacked by the Attacker he is attacking) will disengage his current opponent and engage the new Attacker. Once an Attacker is killed, the Defenders who were attacking that Attacker will choose a new unengaged Attacker if one is available. If no such Attacker is within range, the Defenders will attack any engaged Attacker in range. If no enemies at all are within range, all Defenders return to the Army rally point. If a Defender is killed, he will trigger a respawn to occur at the Army respawn point after a brief recharge period.

VFX

The Defender sword slash animation features a white slash anim trail effect that mimics the sprites of the original game.

Special Attacks

The Holy Order tower spawns Paladin Defenders with a Holy Strike attack. This attack occurs randomly, and involves the Defender striking the ground with his sword. Upon impact, the sword spawns an animated sprite decal, generated in Houdini 15, which expands outward over the ground and applies radial true damage to any enemies it touches.

4.3.4 Dwarven Bombard

Overview

The Dwarven Bombard has a base cost of 125 gold and features a slow ranged attack that does area physical damage and targets only ground enemies. Bombard projectiles are not homing. Area damage is affected the distance of the enemy from the point of impact, and while the tower does not target flying enemies, any flying enemies within range of an explosion will receive damage. The base damage dealt by bombard projectiles is lower than that of mage towers, but is capable of dealing much more damage when used against a tight crowd of enemies. In-game screenshots of Dwarven Bombards can be seen in Figure 4.17.

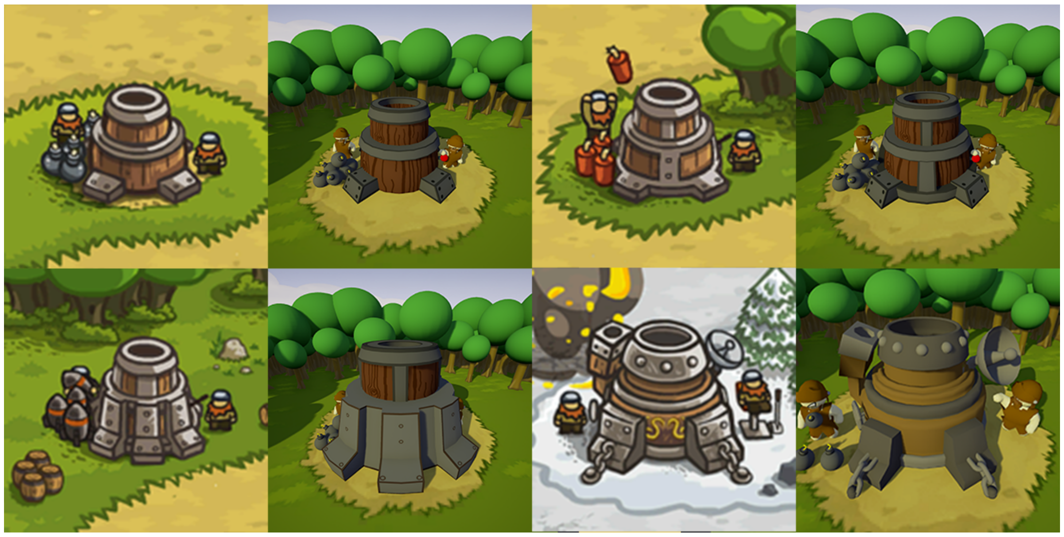


Figure 4.17: Kingdom Rush Bombards and their corresponding Tribute Bombards.

Action Sequence

The Dwarven Bombard is manned by a pair of dwarves, one of whom loads the bombard with bombs while the other pulls the trigger to launch them. When an Attacker comes within range of the tower, the trigger dwarf pulls the trigger and the tower skeletal mesh plays its stylized firing animation. Midway through this animation, a Bomb projectile spawns from within the launch tube, with a horizontal velocity of 20 meters per second and a vertical velocity determined by the predictive aiming function described in section 6.1. Unlike Arrow projectiles, the Bomb projectiles travel a very high arcing mortar path to their target. As these projectiles are not homing, it is possible for them to miss. Once the bomb collides with the floor, it triggers an animated explosion spritesheet, spawns shrapnel pieces, and flashes a brief orange light at the point of impact. All enemies within the blast radius of the Bomb receive radial physical damage.

VFX

While flying, the Bomb projectile emits stylized sparks and smoke puffs, leaving an arc of smoke behind it. The animated explosion sprite sheet is a stylized toon render of a Houdini pyro explosion. The shrapnel from the Bomb is pre-modeled and does not result from a dynamic fracture.

Special Attacks

Kingdom Rush: Tribute does not currently implement the advanced levels of the Bombard tower type.

4.4 Architecture

Due to the iterative nature of the Kingdom Rush games and the limited scope of this project, it is necessary to develop a highly scalable architecture to allow future expansion. To facilitate this approach, an object-oriented, inheritance-based approach is taken to the system design. All classes are developed in UE4's Blueprints system, and inherit from the Actor class or a more specific child class.

4.4.1 Characters

All characters are descendants of the Rusher class, which includes universal variables such as health, walking speed, etc. All enemies are descendants of the Attacker class, while all soldiers and the KRHero are descendants of the Defender class. This class hierarchy can be seen in Figure 4.18.

4.4.2 Towers

All towers are descendants of the Tower class, which includes mesh, character, and targeting functionality. This class hierarchy can be seen in Figure 4.19.

4.4.3 Projectiles

All projectiles are descendants of the Payload class, which includes a static mesh, a collider, an optional trailing particle system, and the amount and type of damage to be dealt. In addition, the Payload class includes a ProjectileMovement component, an Unreal Engine feature that provides basic movement logic used by projectiles, including velocity, gravity, and forward orientation. Forward orientation is used by the Arrow and Missile classes, while the other classes implement a random constant amount of rotation. This class heirarchy can be seen in figure 4.20.

4.4.4 Waves and Spawns

A Kingdom Rush level consists of six to twenty-one waves of enemies, each of which spawns a predetermined set of enemies in sequence. To accommodate this system, Tribute implements a hierarchy of structs for the Wave and Spawn information. The Spawn struct contains essential spawn information such as enemy type, spawn location, spline to travel down, and spawn time offset. The Wave struct contains an array of Spawns and a wave duration. The Wave logic and implementation is handled by an EnemySpawner class, which contains an array of Waves, spawns each Wave's Spawns when the Spawn's time offset is reached, and iterates to the next wave as the level progresses and the current wave's duration expires. A diagram showing the relationships between these classes can be seen in figure 4.21.

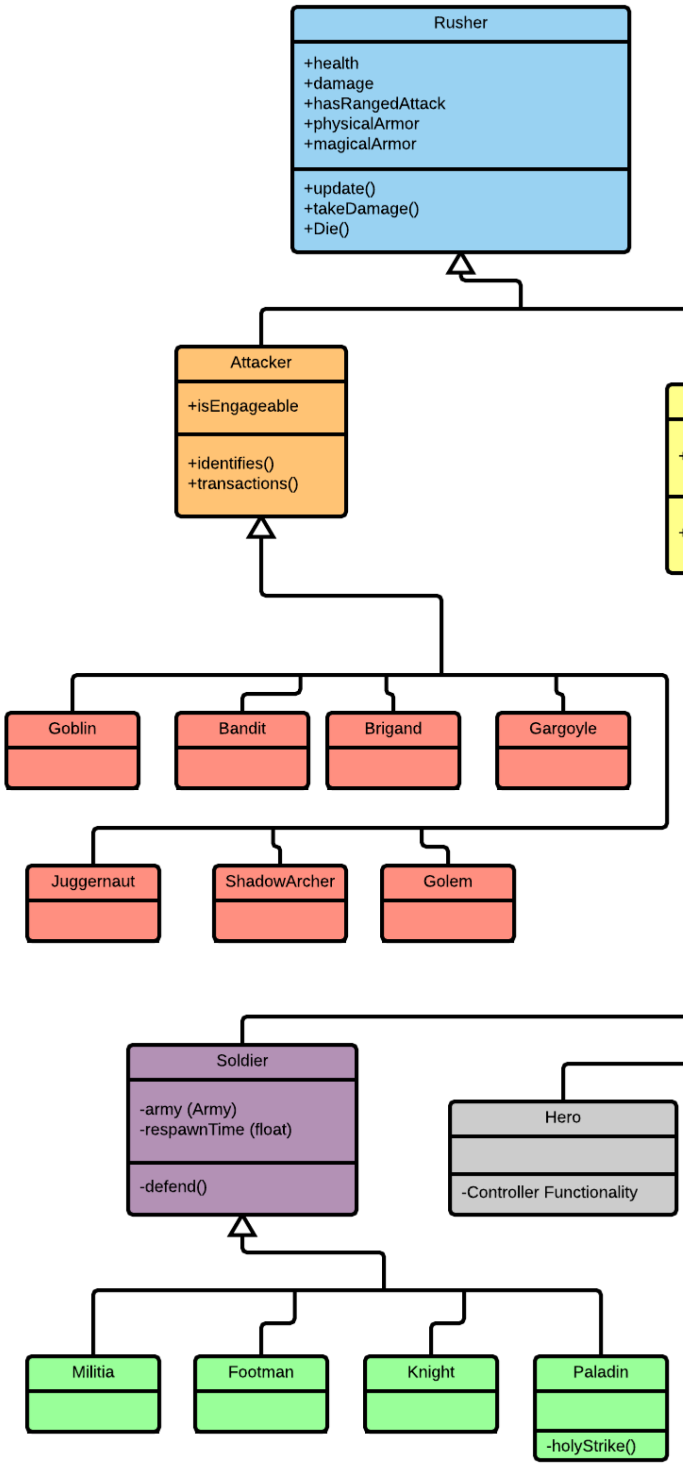


Figure 4.18: The Rusher class and its descendants.

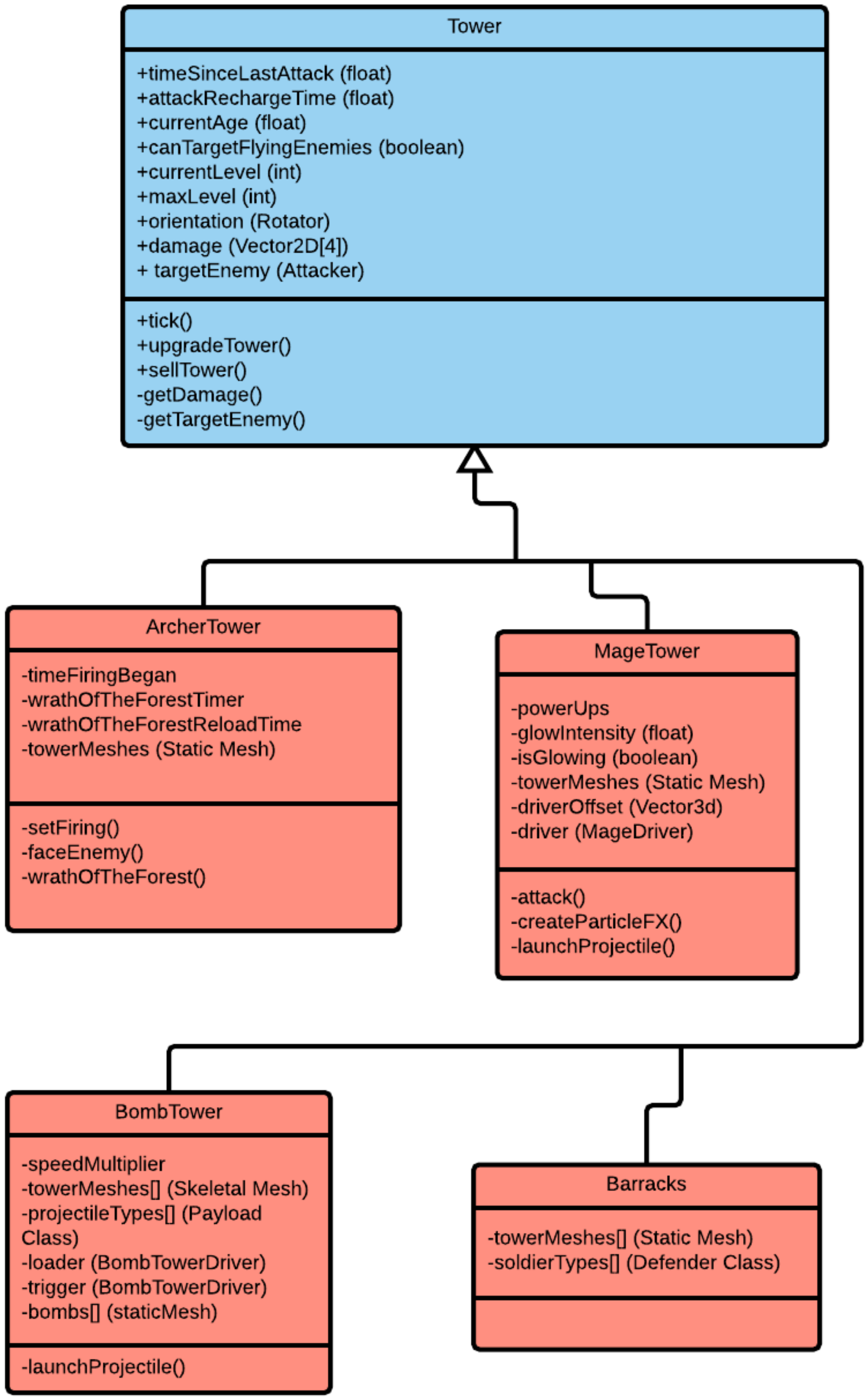


Figure 4.19: The Tower class and its descendants.

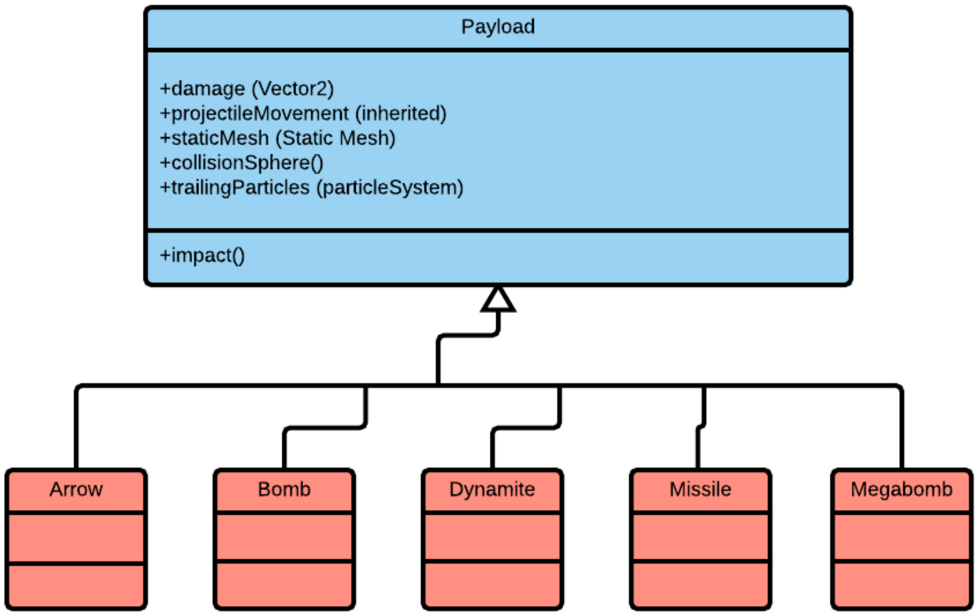

Payload
+damage (Vector2)
+projectileMovement (inherited)
+staticMesh (Static Mesh)
+collisionSphere()
+trailingParticles (particleSystem)
+impact()
Arrow
Bomb
Dynamite
Missile
Megabomb

Figure 4.20: The Payload class and its descendants.

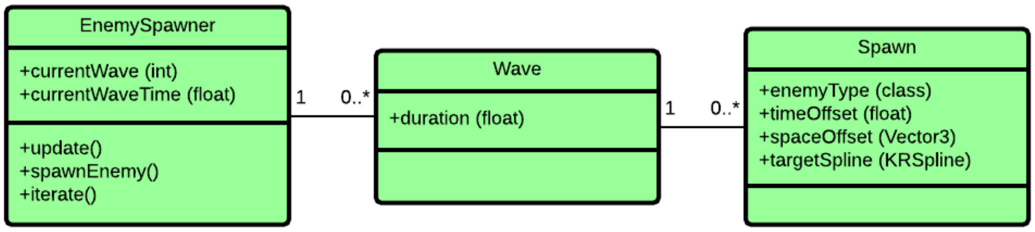


Figure 4.21: The EnemySpawner struct with Wave and Spawn struct member relationships.

Chapter 5

Asset Development

It's a kind of magic.

The Mage

5.1 Environment

5.1.1 Map and Environment Modeling

Environment design for this project is minimal, allowing us to focus on the aesthetic development of the more functional aspects of the game. Where possible, we stay true to the original level design, placing trees and castle walls in similar positions to keep the player in the main field of play. The map floor itself is a flat plane to which we apply the original map level texture. We found this simplistic design to provide less distraction from gameplay than more detailed approaches such as sculpted terrain and painted vegetation, both of which were implemented and discarded. We hope to revisit this portion of the visual design in future work and provide a more visually detailed solution that does not distract the player.

5.1.2 Lighting

Level lighting for this project is also fairly simplistic, and includes two basic lights. The first of these lights is a directional sun light, which signifies a specific position for the sun and creates a visible bright sun in the sky. The second of these lights is a sky light, which captures the appearance of the skydome and uses this information to compute realistic ambient light. To support these effects, Tribute includes a bright blue skydome with simple toon white clouds that was created by modifying the default UE4 skydome in the starter content package with a custom toon cloud created in Photoshop CC.

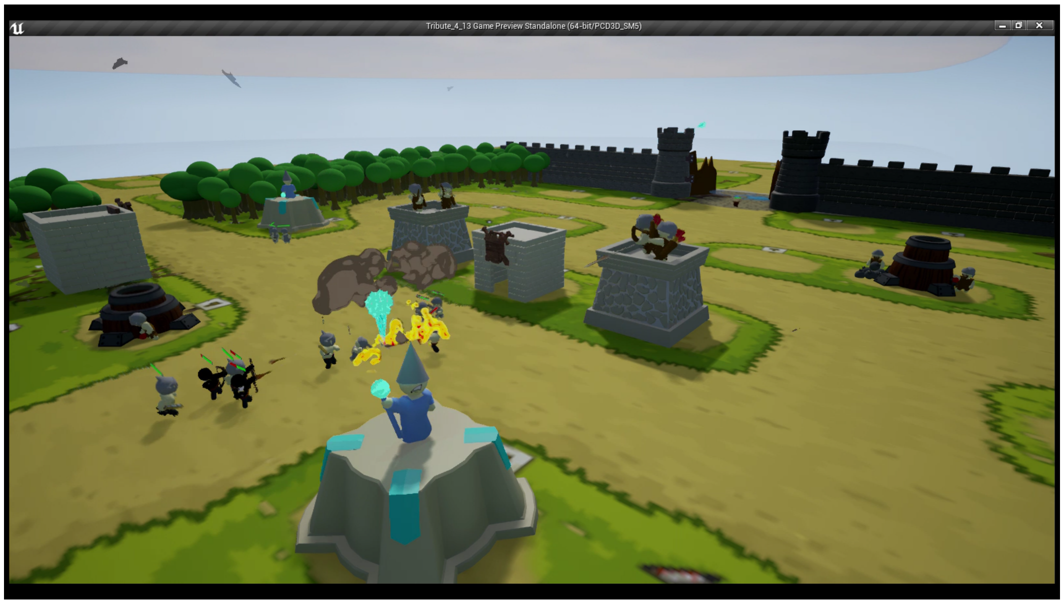


Figure 5.1: Tribute in action.

In addition to these global lighting effects, individual Tribute effects contribute small local lighting. Effects such as exploding bombs and mage bolts make use of point lights, which provide additional colored light in the small area around their respective effects. Mage gems, eplosion sprites, and other effects make use of emmissive materials, which act similarly to point lights by adding glow color to the surrounding geometry.

5.1.3 Sound FX

As Kingdom Rush: Tribute focuses primarily on converting the visual aspects of the original Kingdom Rush game to 3D, the sound used in the 3D game is the same sound used in the original game. All sound FX, music, and dialogue is the property of Ironhide Studios, and no claim is made by us to its authorship.

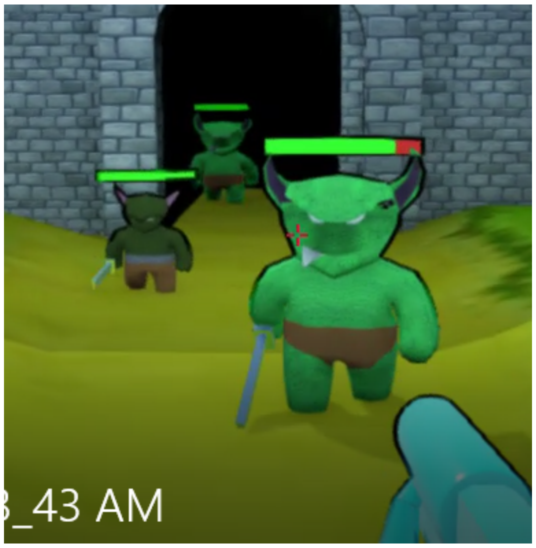


Figure 5.2: An early draft of Kingdom Rush characters.

5.2 Characters

5.2.1 Design and Modeling

As with many other aspects of this project, the most difficult part of developing the characters in Tribute is in exploiting the increased screen space the characters inhabit while maintaining the simplistic, cartoonish look of the original KR characters. An early draft of character development can be seen in Figure 5.2, showing an excessive level of skeletal detail that fails to capture the spirit of the original characters. Characters in the original game seldom consume more space than a 32x32 sprite, providing fairly little detail to design from. As a result, our character design process relied not only on the in-game assets but also on auxiliary material such as game loading and title screens, hero profiles, and the Kingdom Rush comic series. Drawing from this material allowed us to see a more high-resolution version of the original game design, and provided us with such detailed images as those in figure 5.3 from which to base our character design.

While the stylistic choices made for each character varied between models, several decisions remained constant among all characters. These included covering up eyes whenever possible, a lack of facial details such as noses or ears, and stubby arms and legs with no

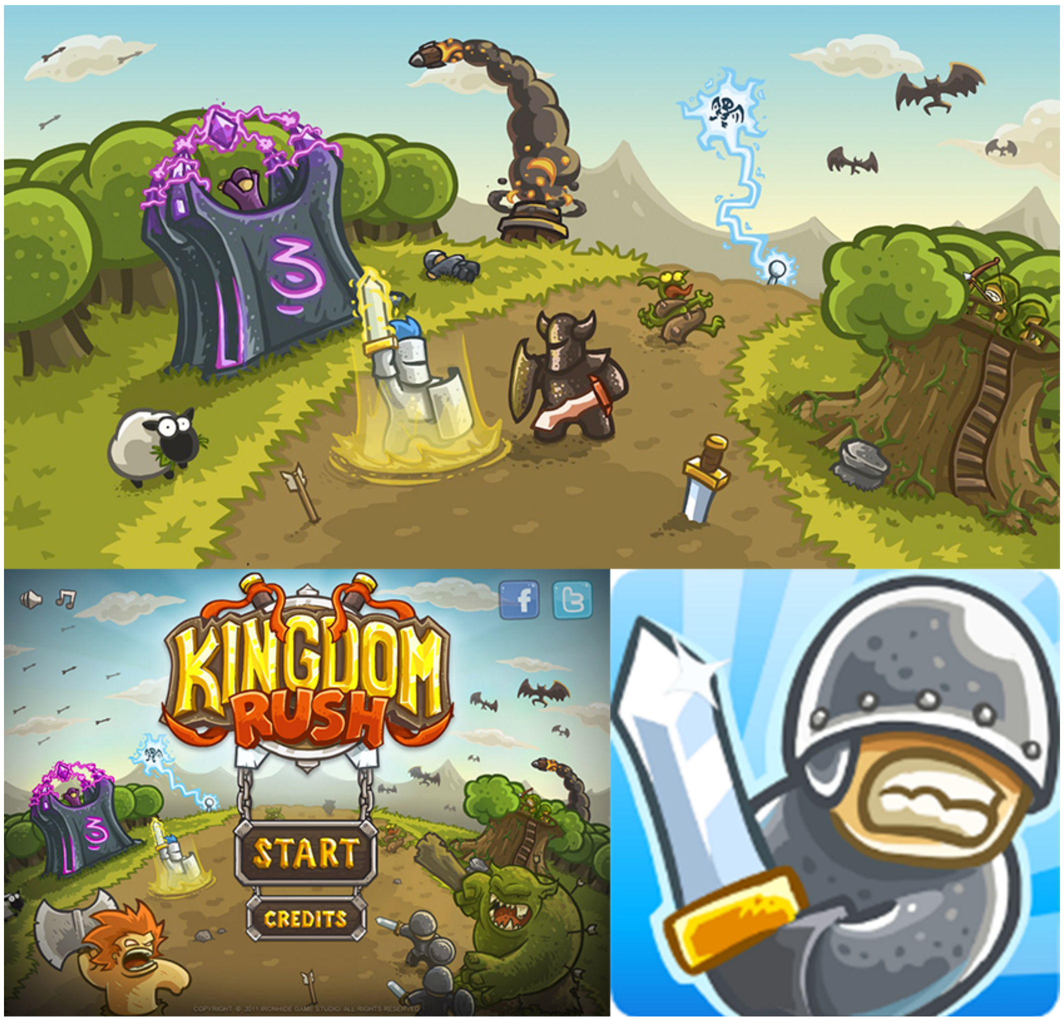


Figure 5.3: Kingdom Rush auxiliary source material.

fingers or toes. Implementing such a simplistic design scheme also allowed us to devote additional time to the rigging and animation of the characters.

5.2.2 Rigging

Models are rigged using the skeletal system in Maya 2016. Where possible, the heat map smooth bind system is used, as this approach greatly simplified the need for further adjustment of bind weights. However, considerable adjustment is still needed, and is accomplished using the paint weights tool. One rig is created for each class of character, with different skeletal meshes created where appropriate. In our implementation, the only shared use of rig was in the Biped rig, which supplied the skeleton for the Bandit, Brigand, Shadow Archer, Militia, Footman, Knight, Paladin, Driver, and KRHero classes. The Gargoyle and Goblin classes were the only classes to make use of their respective rigs, but these rigs could easily be reused in future enemies such as Orcs, Shamans, and Demon Imps. As the Juggernaut and Golem Head rigs were unique enemies, their rigs were not intended for reuse. Joint movement was controlled through a set of NURBS curves which were connected to the skeleton via parent, point, and orientation constraints. IK handles were used to control the motion of arms and legs, with pole vector constraints incorporated to attain appropriate knee and elbow positioning. IK/FK blending was not incorporated in these rigs, as IK alone was found to be more than sufficient to achieve the desired poses. The set of character rigs used in Tribute is displayed in figures 5.7-5.11.

5.2.3 Animation

Animations created for Tribute are based off of the animated spritesheets used by characters in the original game. Studying these original animations was quite effective at allowing us to determine what functionality the characters needed to attain, but the small screen size and relatively low frame count of the animations rendered them unhelpful in terms of providing direct poses to emulate. As a result, the animations were used as general inspiration for their corresponding 3D animations in Tribute. The final animations range from 18-72 frames in length and are looped in the game to create the appearance of continuous motion. For the biped rig, the animations created included idling, walking, fighting, firing arrows, struggling, and dying. The driver characters were also given high and low spellcasting, bomb throwing, and lever pulling sequences. The hero character was given a flag planting animation in addition to making use of the biped animations. The gargoyle rig was given a single flapping animation, as Gargoyles have no other functionality and simply disappear when they are killed. The Juggernaut and Golem Head rigs were given walking, attack, and death animations, and the Juggernaut was also given a rigid body collapse simulation as described in the VFX section.

Animation logic and transition is handled via UE4's Animation Blueprint system. Each character's animation blueprint polls its AnimState variable, updating its own copy accordingly. When a change in animation state is detected, the animation blueprint initiates a transition from the previous animation to the ensuing one. In most cases, animation blending is handled by UE4's default blending function. The exception to this is the KRHero character, which features a custom animation blending developed in UE4's Montage system that allows the character's arms and legs to move independently of one another. This allows the character to run, jump, and stand still while his arms either follow along, fire arrows, or slash with a sword.

5.3 VFX

5.3.1 Rigid Body Dynamics

Rigid body fracture simulations are used in two effects in Tribute. The first of these is in the Rain of Fire spell, which results in several meteors crashing into the ground and spreading hot coals in a circle. While a similar effect could have been achieved with a deferred decal, we believe that a rigid body simulation is a more effective method of displaying this effect. Creating 3D debris for the characters to walk across provides a sense of depth to the effect that could not be created with a deferred decal. To create this effect, we modeled a simple meteor in Houdini 15 and applied a Voronoi fracture to the lower half of the meteor. We then simulated the meteor crashing into a ground plane, applying heavy drag to the fractured pieces to keep them from scattering too far. The resulting simulation was exported as an Alembic file and imported into UE4. When the player casts the Rain of Fire spell, several falling instances of this alembic are spawned high above the map. Once each meteor strikes the ground, the downward motion is paused and the alembic animation is played.

The second rigid body fracture simulation occurs during the death of the Juggernaut boss character. This effect is very similar to the character's death animation in KR, in which the Juggernaut pauses before collapsing into a set of fractured pieces rather than entering

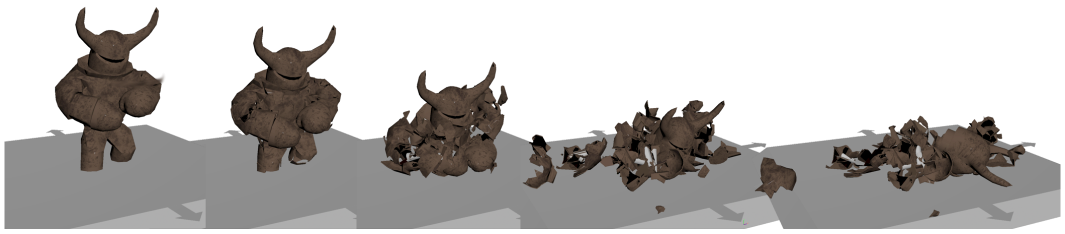


Figure 5.4: The Juggernaut collapses to the ground in a pre-computed Alembic rigid body simulation.

a simple death animation. To create this effect, we imported the posed Juggernaut model into Houdini 15, extruded slightly inward to create thickness on the character, and applied a Voronoi fracture to the character's torso and limbs. The head was left unfractured to create a more dramatic visual. The resulting fractured pieces were simulated collapsing to the ground, and the simulation was exported to an Alembic file. Once imported into UE4, the simulation replaces the Juggernaut's skeletal mesh at the moment of death with no change in position, resulting in a smooth transition to the collapsing pieces. Selected frames from this effect can be seen in Figure 5.4.

5.3.2 Pre-rendered Spritesheets

KR's visual effects are extremely cartoonish and stylized, and it is easiest to imitate this aesthetic by making use of pre-rendered spritesheets whenever possible. Using spritesheets allows us to check every frame of every effect and ensure that the final animation is exactly in line with our vision. As a result, we make heavy use of pre-rendered spritesheets throughout Tribute,

To create the explosive projectile spritesheet, we implemented two pyro simulations in Houdini 15. The first simulation represented the yellow fire, and was given a lower buoyancy and turbulence. The second simulation represented the gray smoke, and was given a higher buoyancy and turbulence. Each simulation was converted to a VDB, remeshed, and given a discrete ramp shader that mapped each range of particle temperatures to an appropriate color tone. When combined, these two simulations appear to form a single explosion. The resulting orthographic render was laid out in a spritesheet and imported into a UE4 shader with a subUV animation. A visual breakdown of this process can be seen in Figure 5.5.

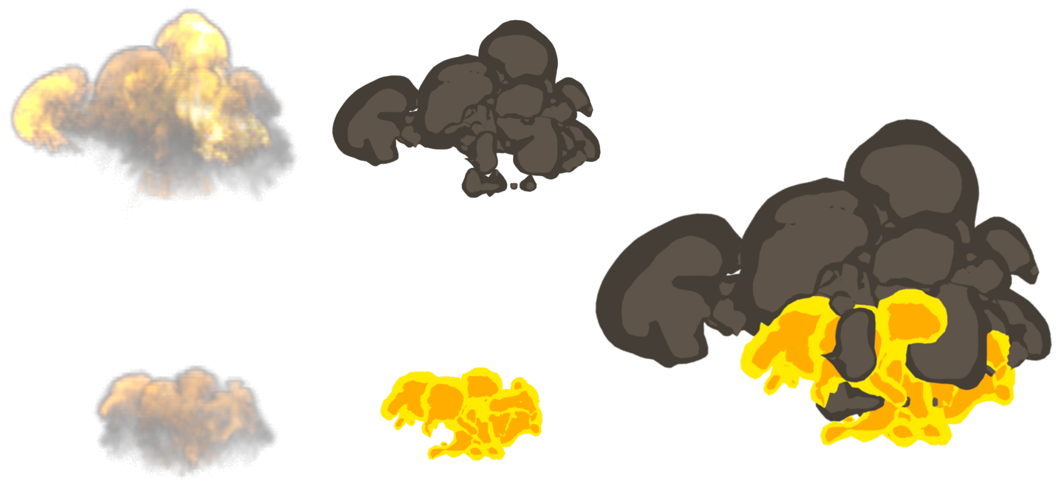


Figure 5.5: Dual Houdini pyro simulations with smoke and fire toon shaders combine to form the final explosion spritesheet.

To create the mage bolt burst effect, we created a radial impulse burst of particles in Houdini 15, converted the resulting particle simulation to a VDB field, and remeshed this field. We then modified the toon pyro shader to accommodate blue and white colors, and applied this shader to the remeshed simulation to achieve our final render. A similar approach was taken to create the Holy Strike blast and arrow impact blood spatter spritesheets, with appropriate changes to particle spawn rates, noise fields, and final toon shaders. A visual breakdown of the blood spatter effect can be seen in Figure 5.6.

A different approach was used to create the mage beam powerup spritesheet. Instead of particles, Houdini geometry was used as a starting point. Several circles were fed through a noise VOP, which varied the position of the individual vertices in the circle based on time. In addition, distinct groups of points were connected by similarly noisy arcs over a longer period of time, resulting in distinct "jumping" arcs of electricity. The circles and arcs were then converted to tubes and rendered from an orthographic view with a purpler toon ramp shader. The resulting render was given a glow effect in Nuke 8 and the final frames were compiled into a spritesheet for use with a UE4 subUV animation.

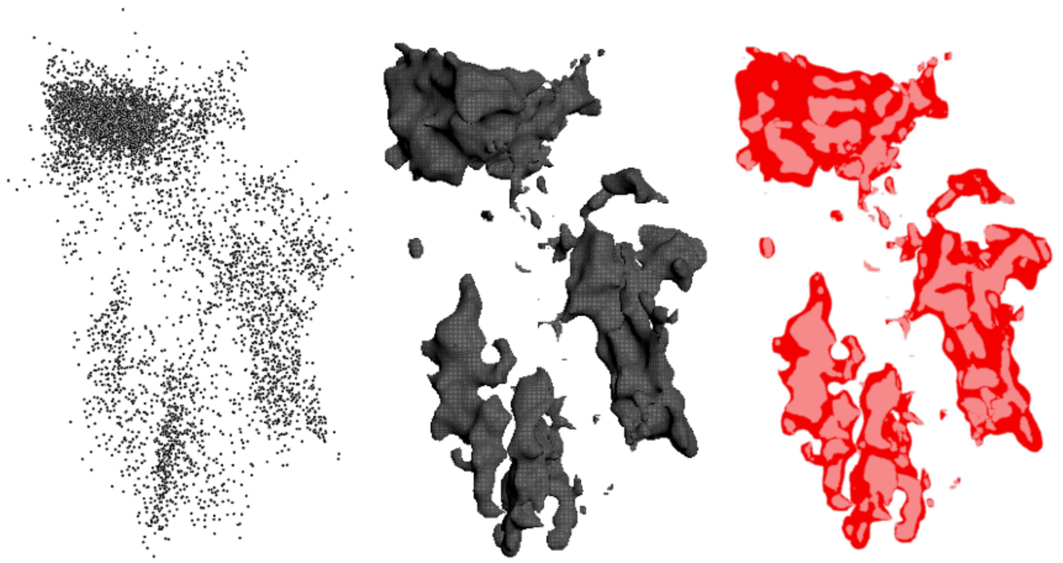


Figure 5.6: Radial burst particle simulation is remeshed and rendered with a toon shader to create the blood spatter spritesheet.

5.3.3 Decals

The art assets in Kingdom Rush consists of two-dimensional sprites, and many of these sprites are used to represent materials and effects which are flat on the ground. Several of these effects have been replicated in Tribute using Unreal Engine's Deferred Decal material mode. Unlike most material modes in Unreal Engine, the deferred decal mode is not applied to a particular piece of geometry or particles. Instead, a deferred decal occupies a portion of 3D space inside the game and projects its color values onto whatever geometries overlap its bounds. This material mode is particularly useful for dynamic effects such as blood spatter, scorch marks, bullet holes, and other effects which appear on relatively flat surfaces. The effects in Tribute which use deferred decals include the blood spatter which occurs when an attacker is killed by explosive damage, the blue spritesheet that appears when a Paladin activates Holy Strike, and the current selection indicator that appears under the player's next target.

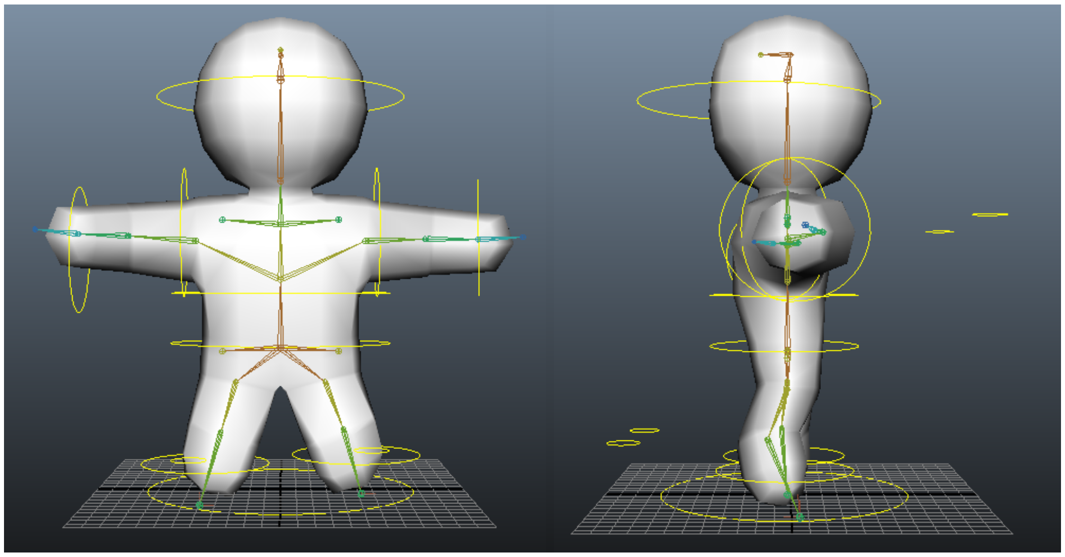


Figure 5.7: The Biped rig.

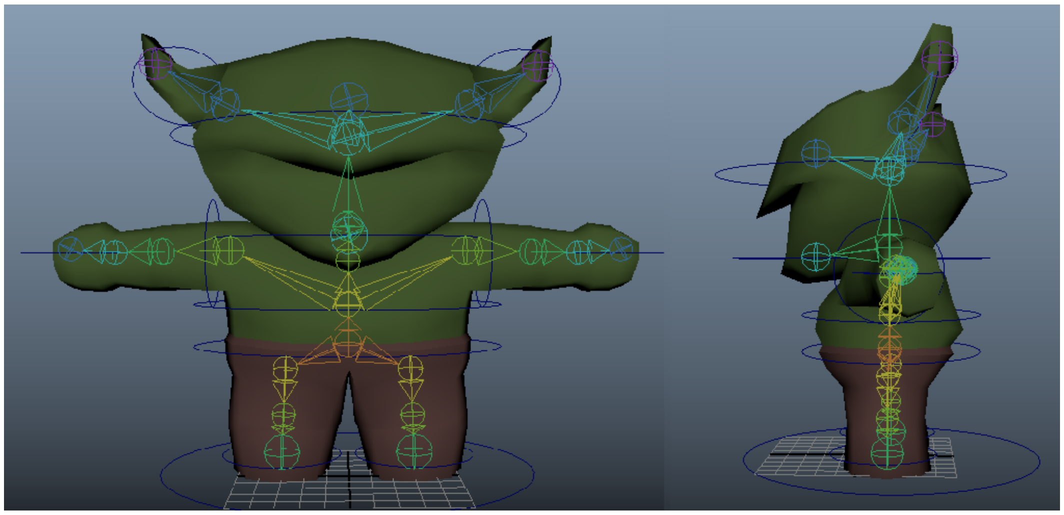


Figure 5.8: The Goblin rig.

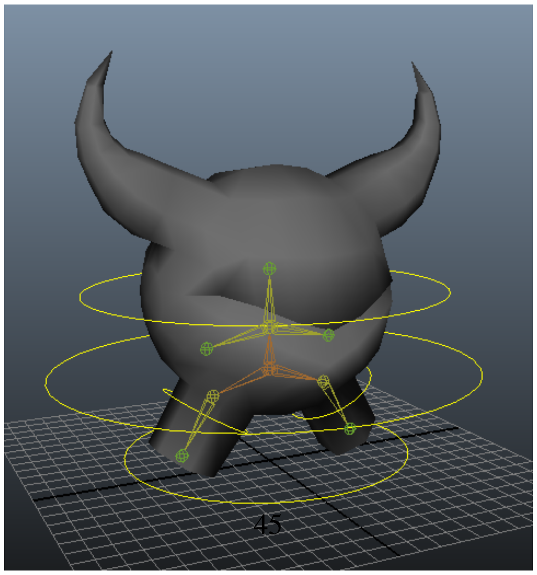


Figure 5.9: The Golem Head rig.

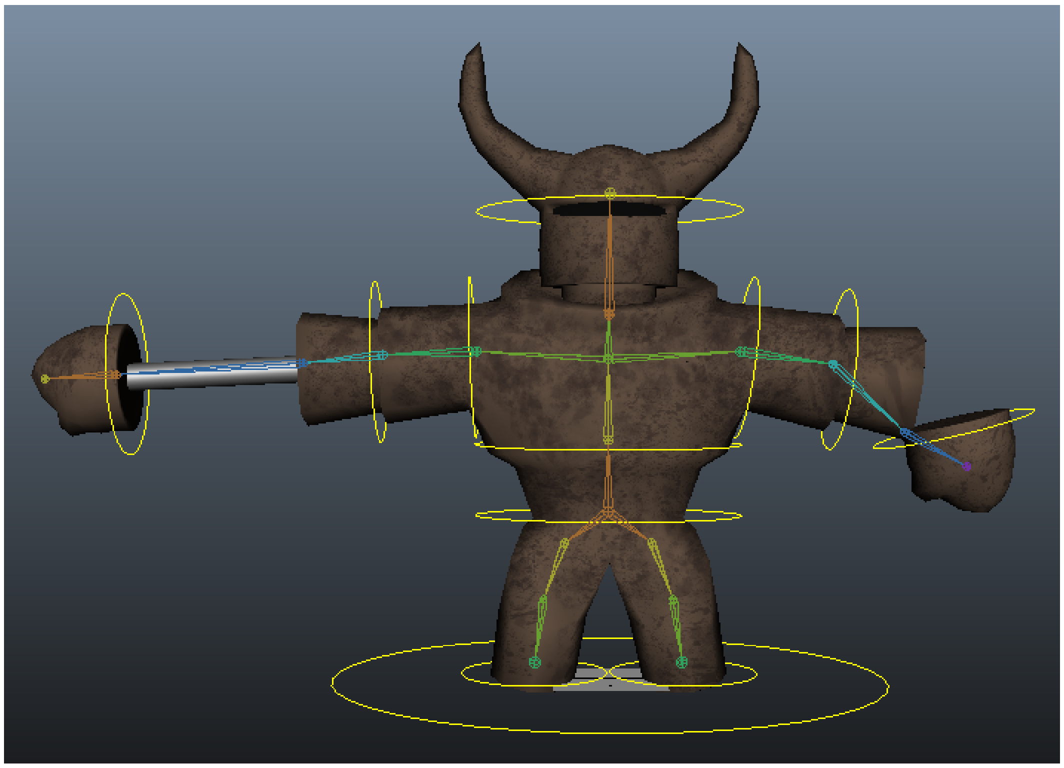


Figure 5.10: The Juggernaut rig.

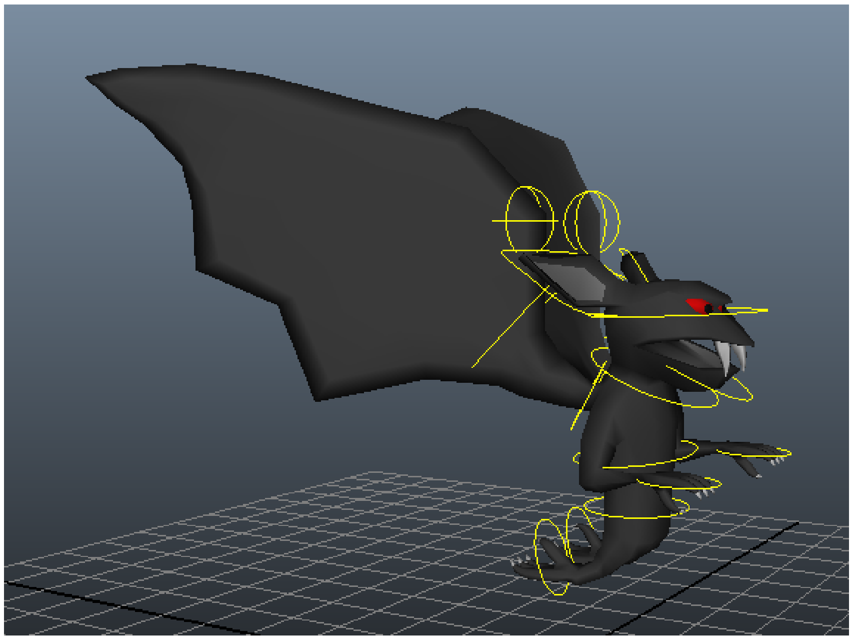


Figure 5.11: The Winged rig.

Chapter 6

New Challenges: 2D to 3D

Mmmh! I don't think that will work!

Vez'nan

Many challenges arose in our port of Kingdom Rush from a 2D top-down tower defense game to a 3D third person shooter; moreover, several of these challenges that arose were unique in that they were created entirely by the dimensional change from 2D to 3D. As a result, solving these challenges was not a question of faithful technical replication, but rather one that required creative design. We describe several of these challenges and their solutions below.

6.1 Predictive Aiming

In Kingdom Rush, most fired projectiles follow an unguided trajectory and do have a chance of missing their target. However, despite their relatively slow speed and interaction with gravity, these projectiles are fairly accurate, due to their implementation of 2D predictive aiming. If the targeted enemy does not significantly alter their velocity after the projectile is fired, the projectile is guaranteed to score a direct hit. This fact is particularly evident while observing fast-moving enemies curving around an archer tower, as the rapid banking of the moving enemies causes each arrow to miss widely. Once the enemies re-enter a straightaway, however, the archer shots return to perfect accuracy.

When replicating this functionality, we first tried to make use of UE4's built-in SuggestProjectileVelocity function. As this function provided no predictive functionality, we attempted to account for the motion of the target enemy by setting the target location to a fixed distance in front of the enemy. While this method proved adequate for explosive projectiles due to their forgiving area damage, we found it to be much less effective when used with arrow projectiles, which only deal damage when scoring a direct hit. To address this issue we found it necessary to implement our own predictive aiming function.

Tribute's implementation of predictive aiming uses an algorithm that calculates the appropriate intercept velocity given a fixed target speed and zero gravity. Once this velocity is calculated, the algorithm adjusts the vertical component of the projectile launch velocity in order to account for gravity. This results in fairly realistic projectile aiming that is guaranteed to hit its target when the enemy is moving at a constant speed in a straight line.

The algorithm used was taken from Kain Shin's blog post ***Predictive Aim Mathematics for AI Targeting*** and was implemented in a UE4 blueprint [4]. The algorithm implements the following system of equations:

$$V_b = V_t + \frac{P_{t_i} - P_{bi}}{t}$$

$$t = \frac{-2DS_t\cos(\theta) \pm \sqrt{(2DS_t\cos(\theta))^2 + 4(S_b^2 - S_t^2)D2}}{2(S_b^2 - S_t^2)}$$

where:

- V_b is the launch velocity of the projectile
- V_t is the velocity of the target at time of launch
- P_{ti} is the position of the target at time of launch
- P_{bi} is the position of the muzzle at time of launch
- D is the distance from the muzzle to the target at time of launch
- S_t is the speed of the target at time of launch
- θ is the angle between the line from the muzzle to the target and the direction of the target's velocity.

This algorithm is utilized by Shadow Archers, The Juggernaut, Bomb Towers, and Archer Towers each time a projectile is fired. While the algorithm is fairly complex, we have found that the relative infrequency of its use does not result in a significant increase in frame calculation time.

6.2 Interaction

The original Flash release of Kingdom Rush is controlled exclusively through a mouse interface. Players can click once on objects to open a relevant menu, and selecting an option from the menu would execute an action and return the player to regular gameplay. Our early implementations of Tribute mimicked this approach, allowing players to open context-based menus by clicking on objects in the screen. However, early playtesting showed that this method created a jarring transition between player control and cursor movement, and players had difficulty switching between the two modes. It became apparent that we would need to develop a more intuitive system for interaction. We developed a more intuitive system that makes use of the player's current location and orientation to determine which object the player is interacting with. This process, and the iterative development process used to create it, is described in more detail in Chapter 7. If the interaction chosen still requires further input, such as when the player approaches a tower, a set of levers appears in 3D space within range of the player, with each lever corresponding to a contextually appropriate action the player can take. Initial playtesting showed that this method of interaction could be a satisfactory replacement for the 2D context menus in Kingdom Rush.

6.3 Pathfollowing

The most challenging aspect of this project involved developing a path following system for the Attackers to use while progressing through the game. While the motion of the sprites in the original game is fairly simple, converting it to 3D posed a unique set of challenges. When the enemies in the original game enter a crowded area of the path, the 2D sprites representing the characters are able to slide past one another without a negative impact on the visual appearance. When ported to 3D, however, the characters are not able to do this, and must resolve the collision either by waiting for the path to clear, taking another path to the goal point, or traveling through the obstacle. A number of methods we considered are described below.

Spline Placement

The simplest and most straightforward method of path movement is to define the position of the character as the location at a percentage of the spline that increases with time. Implementing this method results in very simple and functional movement of the character down the path as desired. However, while it is straightforward to have the character stop and interact with characters in the level, it is less trivial to incorporate avoidance of characters with whom the path-following character is not currently interacting. Our implementation of this method resulted in many characters clipping through each other, greatly detracting from the visual appearance and believability of the game.

AI Navigation

A better option, and one fairly easily implemented, was to make use of Unreal Engine's built-in artificial intelligence pathfinding system to allow the characters to find their own way through the map. This would enable path-following characters to dynamically avoid travelling through other characters and their environment. Our implementation of this system found that if an adequate AI navmesh was created, the path-following enemies performed extremely well, following efficient paths and ultimately reaching their goal through sometimes ingenious ways. However, the behavior exhibited was not a good match for the original Kingdom Rush path-following system. Characters did not stay in the middle of the path, but traveled directly toward the inside corner. Characters also did not stay in formation relative to each other as they do in the original game. Most importantly, characters took the most efficient path to the goal point at all times, a behavior not exhibited by original Kingdom Rush enemies, who make predetermined decisions at road forks that are often inefficient in terms of path optimality. In order to successfully replicate the Kingdom Rush style of play, it was necessary for us to develop an improved solution.

Checkpoints

One proposed solution was to develop a system of checkpoints for Enemies to pathfind to in sequence. By defining a series of checkpoints, we could force the Enemies to not travel directly to the goal point but instead move along the path in a more natural manner. However, multiple weaknesses were found with the system. Functionally, enemies might be unable to reach a certain checkpoint if it was surrounded by other characters, which could result in an enemy circling a checkpoint and traveling backward in an attempt to reach it. This behavior would be completely incorrect. Aesthetically, it would result in enemies suddenly changing direction on the spot rather than gradually turning, a departure from the original game. Rather than implement this solution, we chose to implement the more effective solution described below.

Spline Goal Following

Our final system is a hybrid of the spline placement and AI pathfinding systems. Due to their inheritance of the Pawn class functionality, Tribute enemies are all equipped with AI pathfinding ability, which they make use of to move about the map. However, instead of directing the goal point, each Enemy moves toward a unique Goal actor that moves along a spline. The Goal moves faster than Enemies, but is not allowed to move more than 300 units ahead of the Enemy. If the enemy is engaged by a Defender or encounters an obstacle, the Goal actor will pause its motion until the Enemy finds its way to within the acceptable distance. As a result, the Enemies travel down the path normally, but will make small deviations from the path to avoid any crowded areas. In addition, providing each Goal with a distance offset allows us to arrange groups of enemies in predetermined formation, an iconic feature of KR enemies. The final result is an effective implementation that avoids the

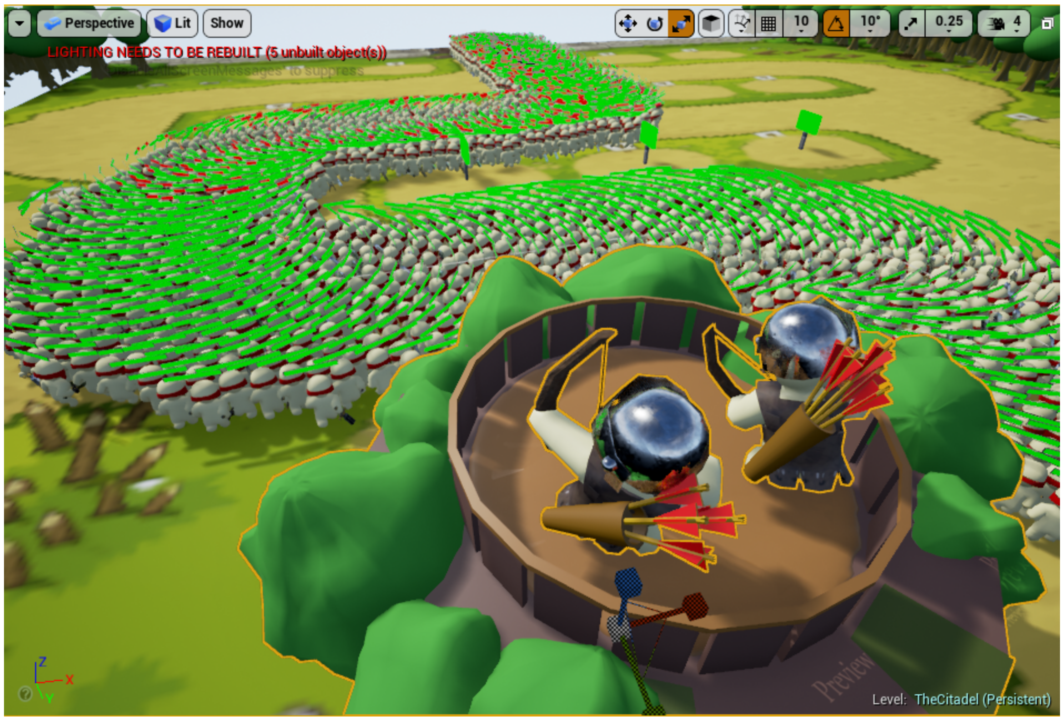

Perspective
Lit
Show
LIGHTING NEEDS TO BE REBUILT (5 unbuilt object(s))
10
10°
0.25
4
Level: TheCitadel (Persistent)

Figure 6.1: 2,000 Bandits fight for space.

problems of the earlier solutions and results in KR style gameplay. This system performed very well in stress testing, allowing over two thousand Bandits to enter the map without leaving the path or stacking up single file, as shown in figure 6.1.

Chapter 7

Playtesting

Back to the shadows of hell!

The Sorcerer

We have designed and implemented a prototype that we call Kingdom Rush: Tribute. This prototype is playable, and can be run on most Windows computers. As a result of the playable nature of the prototype, we were able to recruit a group of gamers who were willing to install and run the game, producing an informal alpha test. After playing the game a few times, each player was asked to complete a brief survey about the game. Existing bugs and glitches were not a focus of this survey. Instead, players were asked to discuss the playability of the game, including what worked and did not work, what was easy to understand and what was confusing, and to provide any suggestions for improving game play. This feedback facilitated several improvements to the basic interactive style of the game.

One player reported dismay about the fact that his character would not continue to attack his chosen enemy after the first blow, but instead had to be re-directed after every attack animation cycle. To address this issue, we implemented an attack-lock system, by which the player's character will continue attacking his chosen enemy until the enemy is dead, the player is dead, or the player disengages by navigating away from the battle. Another player reported confusion when trying to determine which lever or enemy his character would interact with when the mouse was clicked. This player suggested implementing an indicator system to explain which object was the current candidate for interaction. As a result of this feedback, we implemented the current object highlighting system, in which a spinning circle with inward-pointing arrows appears under whatever object the player is facing. When functional, this system was very effective at clarifying confusion for the players, to the point that testers as young as ten years old were able to play the game quite easily with no instructions given.

A common observation, independent of the playability feedback, was that the degree of difficulty of the game was not properly balanced. Players commented that in one version of the game, the player was too weak and that it took too long to manage resources and build towers. In another version, in which the player was given the ability to fire both arrows and missiles, players commented that the hero character was far too strong, and that the game was too easy as a result. This feedback showed us that the game would need to be carefully balanced when finished. In particular, the feedback about resource management, and the increased difficulty involved in having to navigate to a tower's location before interacting with that tower or tower slot, showed us that perfectly duplicating the original Kingdom Rush game balance might not be the ideal solution. Further work will include testing out different variations of enemy strength, player melee strength, and player walking speed in order to address these concerns.

In addition to comments on the playability of the game, players were also asked to give some feedback on the visual fidelity of the game in regards to how well it matched the Kingdom Rush aesthetic. The majority of these responses focused on the use of the low-resolution screenshot of the original map as the game floor, and requested more detail there in addition to the inclusion of houses, sheep, and other props from the original games to increase the sense of immersion. Much of the future work for this game will focus on these aspects.

Chapter 8

Conclusions and Future Work

I see dead people.

The Archers

8.1 Conclusions

8.1.1 Changes in Gameplay

We began this project with the goal of completely replicating the original gameplay of Kingdom Rush in a 3D world. However, as our prototype reached completion, we became aware of several difficulties in reaching this goal that could not be easily addressed. The results of our informal playtesting showed that while the functional aspects of the game had been replicated effectively, the essential changes unique to the 3D aspect of Tribute had made much of the in-game action difficult to observe. Playing on the test map of The Citadel proved much more difficult than anticipated, as enemies entering from alternate paths behind the camera proved difficult to notice, and Golem Heads frequently spawned behind the player, catching him unawares. The new camera's positioning also made it harder for the player to watch his towers in action, and to know when special attacks or player spells were appropriate. Despite these weaknesses, however, players reported greatly enjoying the time spent playing the game, and described new strategies they would not have tried in the original game such as using the player character to distract Shadow Archers so that Dwarven Bombards could attack them more effectively. While we are surprised by the unwanted results that our transition created, we believe that they are natural products of transitioning the game into 3D space. At the same time, we are encouraged by the unexpected positive results, and look forward to further improving the gameplay and overall experience of this project.

8.2 Future Work

8.2.1 Additional Characters

The primary goal of our work entailed the design and implementation of a playable prototype of Kingdom Rush: Tribute. To achieve this goal we focused on bipedal and flying characters. We have not implemented the more difficult Kingdom Rush characters that are characterized by non-bipedal locomotion. These characters include quadrupeds such as Wulves and Worgs, and the many forms of spiders that make up a large percentage of attackers on many levels of the Kingdom Rush series. Since the spiders' ability to quickly lay and hatch eggs mid-wave is one of the most challenging aspects in Kingdom Rush, implementing these characters and the accompanying mechanics is essential to the completion of a full game.

8.2.2 Multiple Levels

Even though this project has focused on the implementation of a single level in sandbox mode, developing a complete set of levels and implementing a system to track player progress is the ultimate goal. Like many video games, Kingdom Rush features a compelling story, and the narrative of an army of honorable defenders journeying through different environments and battling progressively more frightening enemies is fundamental to the Kingdom Rush gameplay experience. While it would not be necessary to develop the same fifteen levels of the original Kingdom Rush game, developing a similar campaign of levels with unique environments and wave structures is the ultimate goal for this project, and one that we hope to achieve in our future work.

8.2.3 Player Progress

In addition to providing support for multiple levels, implementing a player progress tracker would allow for the inclusion of Kingdom Rush's star-based upgrade and achievements systems. At the end of each KR level, the player earns one to three stars, depending on how few enemies make it through the goal gate before the level ends. These stars can be cashed in for a variety of tower and spell upgrades, which range from decreased costs of construction of new towers to unlocked ranged attacks for reinforcements. Linking the purchase of upgrades to the player's progress allows the game to feature more difficult enemies as the player progresses, a common feature in tower defense and other games. In addition to this star system, KR also features an achievement tracking system, which rewards the player for such feats as killing 100 spiders or defeating a certain level without losing any soldiers. Implementing this system is another way in which the original KR experience could be replicated.

Bibliography

[1] John Andrews. Dungeon Defenders Exceeds More than a Quarter of a Million in Sales, 2012.

[2] Trevir Nath. Gaming will hit $91.5 billion this year - Newzoo, 2016. http://www.nasdaq.com/article/investing-in-video-games-this-industry-pulls-in-more-revenue-than-movies-music-cm634585.

[3] BBC News. Casual games make a serious impact, 2017. http://news.bbc.co.uk/2/hi/technology/7301374.stm.

[4] Kain Shin. Predictive Aim Mathematics for AI Targeting, 2009. http://www.gamasutra.com/blogs/KainShin/20090515/83954/Predictive_Aim_Mathematics_for_AI_Targeting.php.

[5] Brendan Sinclair. Investing in Video Games: This Industry Pulls In More Revenue Than Movies, Music, 2017. http://www.gamesindustry.biz/articles/2015-04-22-gaming-will-hit-usd91-5-billion-this-year-newzoo.

[6] Alexander Sliwinski. "Dungeon Defenders picks up gold from 600K sales, 2013.

[7] Tower defense, 2017. https://en.wikipedia.org/wiki/Tower_defense#A_new_breed_of_3D_games.

[8] Luis Wong. Making Kingdom Rush: an Uruguyan Tale, 2014. http://www.polygon.com/features/2014/9/10/6045757/kingdom-rush-uruguay.