March 2011

Chronopigger: The Making of an IGF Winner

Edward P. Orsi
Worcester Polytechnic Institute

Elena Estelle Ainley
Worcester Polytechnic Institute

Matthew R. Lyon
Worcester Polytechnic Institute

Morgan Quirk
Worcester Polytechnic Institute

Thomas Kangchao Liu
Worcester Polytechnic Institute

Project Number: DMO-4419

Chronopigger: The Making of an IGF Winner

A Major Qualifying Project
submitted to the Faculty
of the
WORCESTER POLYTECHNIC INSTITUTE
in partial fulfillment of the requirements for the
Degree of Bachelor of Science
by

Elena Ainley

Thomas Liu

Matthew Lyon

Edward Orsi

Morgan Quirk

Date: March 4, 2011

Approved:

Professor Dean O'Donnell, Major Advisor

Professor Gary Pollice, Co-Advisor

Abstract

The overall goal of this project is to deliver a standout video game eligible for submission to the Independent Games Festival Student Showcase. To that end, we designed and implemented *Chronopigger*, a 2D platformer as well as an accompanying level editor, both of which were built on top of the XNA framework. The game combines physics simulation and time manipulation with a cartoony, vector art style and an Amish farmer in search of his favorite pig.

Table of Contents

Table of Figures

Background

Independent Games Festival

The Independent Games Festival, or IGF, seeks to be the "Sundance" of the gaming world and from the onset of this project, the goal was to develop a game capable of being a finalist in its Student Showcase. The IGF is an annual celebration of the most innovative and compelling games made by independent studios that year. While there are other juried festivals specifically for independent games, such as IndieCade, PAX10, the Slamdance Games Festival, and SXSW Screenburn, the IGF is both the longest running and largest of the competitions. Taking place during the Game Developers Conference, finalists have an opportunity to showcase their games to thousands of developers.

The Student Showcase is a special category reserved for games made by students still completing their degrees. In the past, teams that won the Student Showcase received not only the prize money, but more importantly incredible exposure and networking opportunities. Over the past five years, Valve Corporation has hired two winning teams, those that made *Narbacular Drop* and *Tag: The Power of Paint* to work on their award-winning *Portal* series (which itself is based on *Narbacular Drop*). Additionally, a number of the student finalists have gone on to publish their games as commercial titles, including *The Misadventures of P.B. Winterbottom* and *And Yet it Moves*. We felt that, while lofty, our goal of making a Student Showcase finalist would be both worth the effort as well as an achievement well within our grasp.

IGF Research

With this goal in mind the team decided to research both the competition and the characteristics of the winners. As a group, we played each of the 2010 finalists listed on the IGF website, assuming there was a publicly available demo or game. We also played through many of the games from years prior to 2010; however we gave them a lower relevance with regards to the current state of the

IGF competition. In the event the team did not have access to a copy of the game, we examined marketing media, gameplay screenshots, and video. By paying close attention to design, technical, and artistic decisions, trends in successful IGF games began to emerge.

Generally the IGF judges tended to favor games with novel visual styles, simple controls, and interesting technical challenges. More importantly than a simply original artistic style, the graphics of the highest performing games were overwhelmingly 2D. Within the category of 2D, games that relied on traditional platforming elements employed to create novel gameplay experiences through complexity enjoyed massive success. Prominent examples of this trend include *Continuity,* the winner of the 2010 IGF Student Showcase, and *Paper Cakes*, a 2010 Student Showcase finalist.

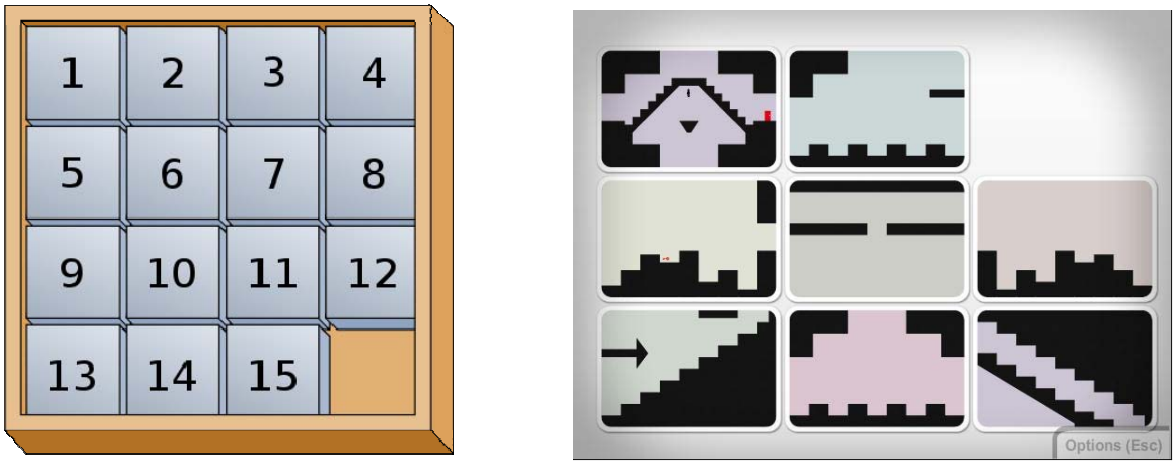


Figure 1 – Sliding puzzles (15-Puzzle, 2006) combined with platforming elements yields *Continuity*'s award-winning gameplay (Continuity, 2010).

Continuity combined traditional 2D platforming with the classic sliding puzzle. In doing so, they created a game in which the player must shift around chunks of the world in order to be able to progress within a level. They chose a minimalistic style, using only pastels and silhouettes to bring their world to life. Their minimalist style ensured that the main focus of the game was on the mechanic rather than flashy graphics.

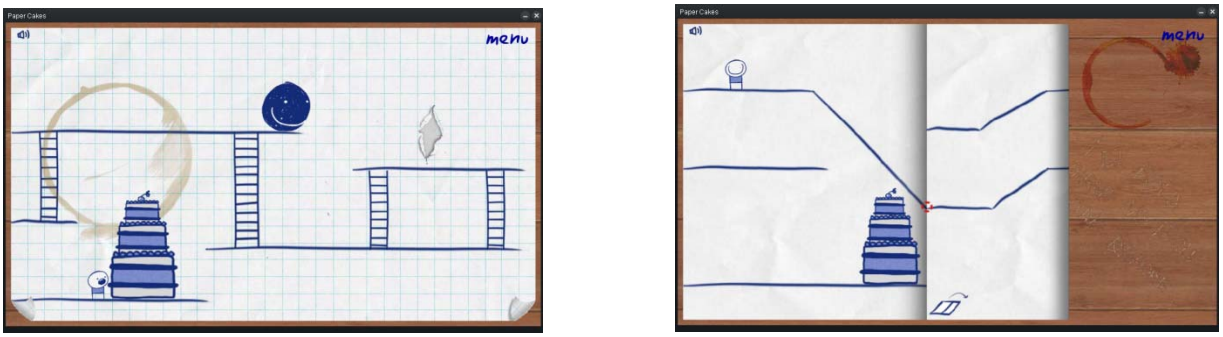


Figure 2 – Examples (Paper Cakes Screen 3, 2010) (Paper Cakes Screen 2, 2010) of *Paper Cakes*' unique art style and mechanic

Paper Cakes took a slightly different approach, of which their art style was an important factor. The game opted for a hand-drawn style for its characters and levels assets. Various types of paper provided the backdrop the levels occurred on, complete with smudges and coffee stains, further reinforcing the notion that the game could simply be a doodle on a piece of paper. The primary mechanic the game employed was the ability to fold the paper over on itself, revealing new platforming paths and allowing the player to access new parts of the level. The art style provided the proper affordances to show the player how the game should play.

Each of these games also sought to differentiate themselves visually through both carefully chosen color palettes and polished, consistence presentation. The technical complexity of the finalist games varied but did notably focus on serving the gameplay and overall style of the game.

Brainstorming the High Concept

Taking this initial research, the team began brainstorming various ideas. An important facet of these early brainstorming sessions was that the team did not become beholden to any of the ideas that come out of them. While ideas that seemed promising were to be explored in some depth, the sessions were as much for culling weak ideas as generating strong ones.

The first game idea the team explored revolved around an arsonist using fire-starting abilities to complete straightforward platforming game. As the player progressed a plot twist would reveal the arsonist was reliving memories and would be forced to witness or confront the psychological cause of his or her desire to light fires. With this plot point revealed, the player attempted to escape from the mental trauma of the dream by retracing the steps to the reveal. In this play through the game levels would bear the damage that the player had wrought as burn holes in a canvas or paper on which the game is played. The team worked on this concept in the early stages of the project, including producing prototypes of the mechanic and rough concepts of the visual style. Despite the strong narrative and art focus, the arson game idea was eventually dropped due to the difficulty of finding captivating or interesting gameplay. The IGF research had shown that while intriguing narrative and art style were both elements of finalist quality games, compelling gameplay was the most important aspect of a submission. Thus, lacking that core element, the team abandoned the game and renewed its brainstorming efforts.

While still considering a platforming centric 2D game, the team broadened its search interesting or novel game mechanics beyond two dimensions. The team quickly designed a 3D game which revolved around the mechanic of time being frozen. The player would encounter dramatic events that are completely frozen in time, such as an earthquake causing a massive landslide. The player would be unable to directly interact with the world, but he would have control over which time during the event

he was currently looking at. Thus the player might start off looking at the aftermath of the earthquake, with the rubble of destroyed buildings surrounding him. Then, the player could rewind time to an earlier state, perhaps to before the earthquake struck, when the buildings were still intact. The gameplay would revolve around the player jumping to specific key-frames during these events in order to be able to platform around. This idea was eventually scrapped in part, as it really centered on 3D gameplay, and the team wanted to pursue a 2D game. Additionally, the team felt it would be difficult to generate 3D assets that would deform and decompose into smaller pieces in order to fully realize building-level destruction. However, the general theme of the player being incapable of altering his environment except by jumping to different states resonated with the team. We felt it created a sense of isolation and loneliness in that while the player can re-watch these events a number of times, he can never change them.

Building off of this theme, we wanted to juxtapose the time manipulation mechanics with a main character that would never have considered time manipulation to be a plausible reality. *Chronopigger*'s story evolved out of a desire to keep the tone of the game light hearted yet meaningful and non-trivial. This design intent fundamentally shaped the team's approach to all aspects of the final version of the game.

Refining the Concept

Figure 3 – Gideon's Early Style

With an overarching design decided, the most important task was topical research. The main character of the game was conceived as a hybrid between a Luddite and an Amish farmer. Drawing on elements of Amish culture for inspiration, the first designs of "Gideon" evolved. Extensive multimedia background research into Amish culture yielded inspiration for both story and visual elements (The Story of The Hex Sign, 2010). From watching *Witness* (Weir, 1985) to researching the commercialization of Amish cultural products, a picture of a real character began to form. The overall tone of the game aims to be humorous and slightly silly while still creating a meaningful story. Cast as a simple farmer out to sell his products, Gideon must ironically use the very advanced machinery he so despises. Aiming only to save his soon-to-be-prized pig, Amos, the heroic farmer must utilize his bravery and simple ingenuity. All of these backstory elements led to the creation of a main character with a serious attitude and a plan to save his prize. A disaster-ridden and torn landscape organically evolved as a foil to the straight-laced nature of Gideon. The environments and levels that the player plays through are infused with contradicting safe aesthetic and chaos.

Figure 4 – Menacing level design in *Chronopigger*

Design Choices

The artistic style of the game evolved to match the tone and gameplay. The team worked with a fellow WPI student, Ryan Chadwick, to develop concept art for the style and feel of the game. Tasked with sketching out scenery and objects the player would encounter, Ryan delivered excellent renderings that would later inspire both level design and gameplay. This concept art provided a launching point for the artistic iterations throughout the game's development and inspired the first style the team chose.

Figure 5 – Scene Concept Art

Figure 6 – Object Concept Art

Out of the initial brainstorming sessions a simple, polygon-based style overlaid with gunge patterns emerged. This style attempts to mimic paper based cartoon animation with a rough, hand-made look. Early in development the style was chosen for its various merits, primarily the striking, handcrafted visual possibilities. In addition to making the game look unique, the papercut method provided a clear asset creation pipeline composed of simple shapes with basic shading created outside of the game. After these assets were finished the grunge and paper overlays are applied in-game to achieve the effect. This decision to use polygons rather than tiles allows the art assets to both be modified quickly and take on more complex shapes.

Figure 8 – Example of "Papercut" Style

Figure 7 – Example of Vector Style

As production progressed the team realized the art was not matching the gameplay or the story as well as it initially seemed. The backstory of the game shifted and condensed into a tale of a farmer trying to rescue his precious pig from certain doom with the graphics needing a reboot to match. The grunge overlay effect consumed more time to implement than anticipated which motivated the team to search for a new style. After another round of brainstorming and sketching a style based on vectors emerged. Vector based art allowed the team to scale elements or entire assets quickly to suit the game design. This new process was continually refined throughout the development of the game to achieve a style that was soft, friendly, and slightly silly. In addition to the overall aesthetic changes the main character's size and proportions were altered. This change affected several other art assets which needed not only a stylistic overhaul but also adjusted proportions or even features. Ultimately the transformation from serious to silly and stark to cartoon consumed much of the art team's time but it yielded a more powerful and engaging style.

Gameplay Description

Chronopigger, in its current iteration, is a 2D puzzle-platformer whose core gameplay centers on manipulating time to alter the environment. The primary mechanic driving the game is the ability to step through pre-recorded key frames of how an event played out. The player can only move when they aren't shifting time and vice versa. Additionally, players cannot move objects into their character; the

game simply prevents them from doing so. This mechanic creates situations in which the player must shift time in order to move obstructions that are blocking a path or move platforms into place to create a path. The secondary mechanics we layered on top of the base time manipulation are referred to as “time glue” and “time loop”.

Time glue freezes an object in place, preventing it from moving even when the player shifts time. Combined with the primary mechanic, this mechanic allows players to stand on an object and shift time, without worrying about being prevented from shifting time due to an object colliding with them. Time glue’s primary purpose is to allow players to freeze platforms in place to enable platforming. Two objects may be falling at different heights; time glue allows players to freeze one object and then progress time until the other object is at the same height to enable players to get across a gap.

Time loop causes an object to oscillate through a few seconds of its timeline, even when time is paused. By time looping an object, the player creates a moving platform that enables access to previously unreachable areas. When combined with time glue, time loop allowed us to create puzzles in which a player would time loop objects to create elevators that enabled players to reach an object that needed to be time glued in order to be able to progress forward. Similarly, through clever use of time loop and time glue, players could loop objects then glue them in their new locations, granting players a limited ability to rearrange the scene.

These three mechanics, combined with the aforementioned quirky art style, form the current state of *Chronopigger*. Comparing the mechanics and art style to the prior finalists we examined during our IGF research, we felt that this combination was a suitable starting point for developing a game with a high potential to be a finalist candidate.

Technical Methodology

This section covers the processes and concepts used in the development of the technical features in Chronopigger. This includes software engineering, editor features and details on the process.

Originally we had planned to use iterative development, keeping track of our tasks on Pivotal Tracker, but it quickly fell apart. Each week we set out to complete certain tasks but the team would quickly split and begin working on other features that we felt were more important at the time. Because of a lack of clear weekly goals, we collapsed in a waterfall development process, and in the end we were left without much time for testing.

Design Patterns

Designs patterns are general reusable solutions to commonly occurring problems in Computer Science. We ran into several of these common problems while working on *Chronopigger* and used design patterns to properly engineer clean solutions to these issues.

A useful pattern for many games is the concept of Object Pooling. The idea of this pattern is that creation and deletion of new objects is costly, so doing it more than necessary is to be avoided. In C# we also have to worry about unpredictable garbage collection at runtime, so it seemed especially important if we were going to be creating and destroying objects frequently. The only place where this is important in our code is for particle effects. The particle effects class acts like any other Sprite Group (collection of image objects) but it creates all the objects it will need once, at the beginning of the level. In the case of the jump cloud particles, it creates 50 particles during the loading screen, and “spawns” objects when needed by just making them visible. When the lifetime of the particle expires, it turns invisible again and is flagged for reuse.

The Singleton pattern is useful when you want to restrict an object to being instantiated only once. We desired a global Game Manager class, and we used a singleton to do this. The Singleton pattern allowed us to have a global class that we could reference for important game related state.

We use the Iterator pattern to walk lists throughout our code, as opposed to looping through the indices. In C#, this means we automatically protect our lists from being modified while iterating over them, which has prevented several bugs. Additionally, it allows us to switch which kind of collection we're using without modifying the code that iterates over it.

We followed the command pattern when implementing recording events. Essentially, when rewinding or fast-forwarding time, the player was committing a series of transactions that impacted the state of the world. These transactions needed to be undoable so that the player could go forward / backward in time at will. Thus all actions that occurred in our game were serialized into self-contained recording events in order to facilitate this needed functionality.

Refactoring

Throughout the process of developing *Chronopigger* we periodically refactored our codebase. Most of the time, we made small changes to make the code more reusable and extensible, including rearranging classes, fixing prototype code, renaming things, and so on. We did some major refactoring between B and C term, where we created a tools system for the editor and fixed the main game state machine. This was very useful in terms of making the code more readable and less buggy, but it also introduced some new problems because certain features weren't re-implemented. For example, the editor was unable to load certain event types after the refactor because the tools weren't finished being implemented, despite their old hard-coded versions having been deleted.

This could have been avoided by using test driven development and having a good set of unit tests. Unit tests would have created an environment much less prone to buggy code. Having unit tests

would have forced us to cover all our ground while refactoring, so that the above mentioned mistakes would have occurred.

Code Walk

In the middle of C-term, we did a code walk with Prof. Pollice, our Computer Science advisor. This consisted of us showing him diagrams of our code structure and displaying samples of our class hierarchy and implementation code. He gave us some feedback about patterns to try to utilize, such as the Factory and Flyweight patterns. We also brainstormed with him some solutions to problems we'd been having with the editor code. The code walk did prove to be useful and enlightening, but occurred too late for us to make many of the relevant changes. It would have been better for us to do a code walk early in the design phase, and then a later one part way through implementing the game logic.

Architecture / XNA

Early on, we decided to use XNA as our game framework. This decision was informed by an analysis of the types of games that win IGF and the requirements for our specific design. XNA meant that we would have to write our own custom game engine on top of the framework, allowing us to structure the whole system around the time mechanics. Using an existing game engine could have made it difficult to engineer *Chronopigger* because the engine's creator had more traditional games in mind. For instance, an engine like Flixel includes features like a flexible sprite system with animations, but it would be difficult to add time-recording functionality to Flixel's sprite class, since it is written with a particular purpose in mind. Designing our own engine let us couple physics and sprites together, making it much easier to engineer time mechanics, albeit slightly less reusable.

In designing 2D games, there are a number of decisions that must be made early on because they affect multiple pieces of the implementation. For *Chronopigger*, one main decision was the use of polygon-based collisions and levels instead of using tiles. After brainstorming and discussing the options,

it became clear that polygons would be the right answer because it allowed for more diverse art assets and level design options. Originally, we wanted to have “set pieces or image/polygon groups that would be duplicated and placed multiple times within a level. This would reduce the number of assets required to build levels for the game. However, the feature was never implemented because we wanted each part of the level to have custom art. In the end, each moving polygon in each level has an art asset placed on it, and there is a single asset for all of the static ones.

Another decision we had to make at the beginning of development was how to handle the physics. We knew it would be cleaner and more intuitive to work within a custom physics engine which had the ability to simulate both forward and backwards, but the time required to implement our own physics was going to be prohibitive. Instead, we used a popular 2D physics engine called Box2D, and built our engine around a recording system instead of trying to run the simulation in reverse. This also gave us the potential to have many physics events happening at once without the game slowing down.

Editor

Early on it became clear that we would need a homebrew solution for level creation. While other 2D level editors existed, we had the unique requirement of needing to set both the location and time of objects in our levels. The available editors offered the ability to place assets and environments where we pleased, but none offered the ability to set timing information for assets. As such we engineered our own editor, using the combination of XNA and Box2D.

We began with a prototype implementation of the editor. This initial version consisted of the minimum necessary functionality in order to develop levels. The focus was on having functional software rather than spending time on design. While this allowed us to develop the first level quickly for our proof of concept, it quickly became apparent that the editor, in its current state, would not be usable in the long-term. A list of the features presently available in the editor was constructed and then

a new architecture was designed around both the existing functionality and the desired features that may have been implemented in the future.

A state-based approach was chosen wherein the editor could be in any of a number of “tool” states at a given time. Rather than hard-coding everything into the editor itself, functionality was abstracted behind a tool interface that allowed new tools to easily be added, greatly increasing the extensibility of the code. Under the new model, the only modification adding a new tool required was registering the tool with an associated key press through a tool manager class. This registration step had the added benefit of ensuring that two different tools were incapable of being bound to the same key.

Art Methodology

This section describes the pipeline and workflow for moving from puzzle concept to full level in game, complete with art assets. We briefly outline the various stages that a concept evolved through as well as the work necessary to progress from stage to stage.

Puzzle Design

After the high concepts for the levels were decided upon, we would schedule meetings to determine the puzzles that would appear in each level. These meetings usually consisted of us using a whiteboard and markers to display ideas and rough sketches of the puzzles.

Rough Mock-Up

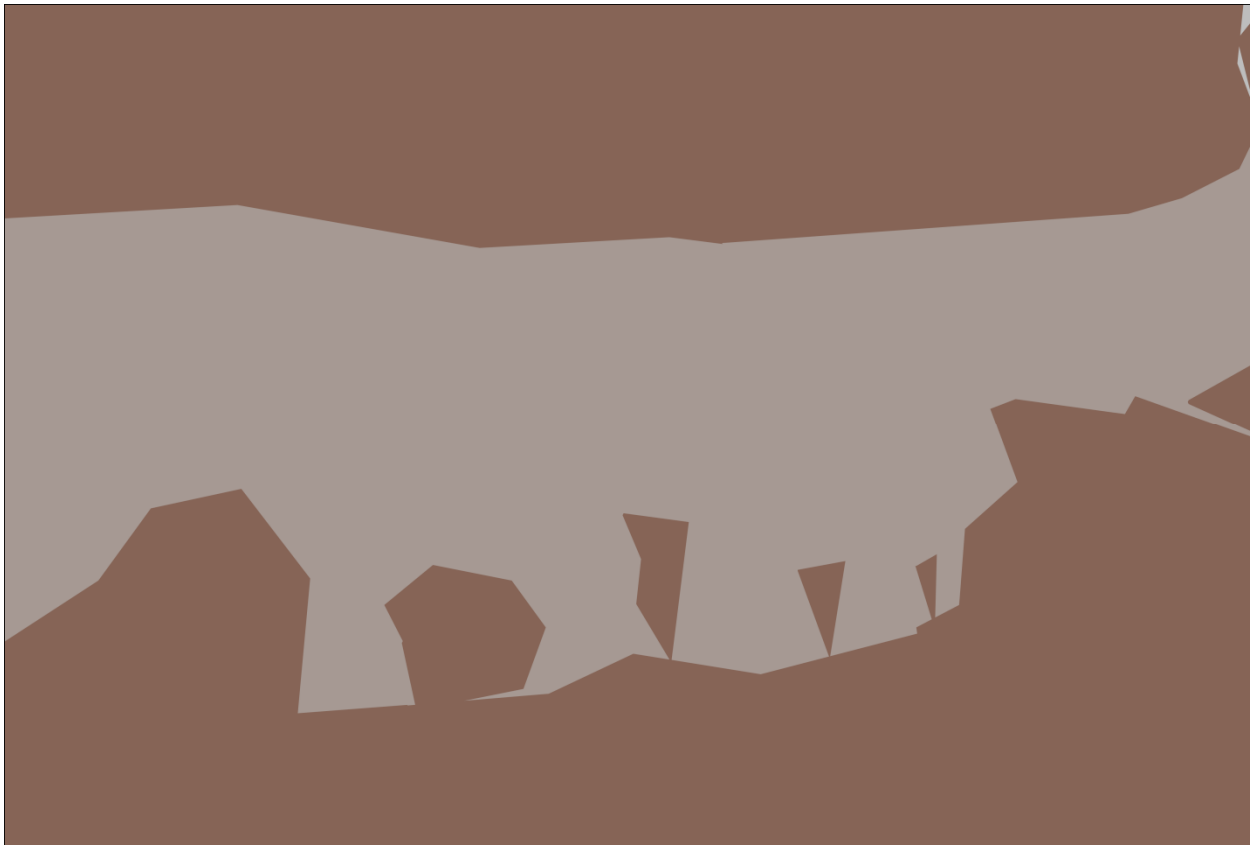


Figure 9 – Rough Photoshop mockup of a puzzle from level 1. The player needs to rewind time in order to hop across the falling stalactites to get across the gap.

Once these puzzles were in place, the Tech team would quickly copy down the idea onto a sheet of paper and would make a rough mock up in Photoshop of where the walls and the ground of the level would be. This would also help determine the size of the overall level.

Creating Polygons with Level Editor

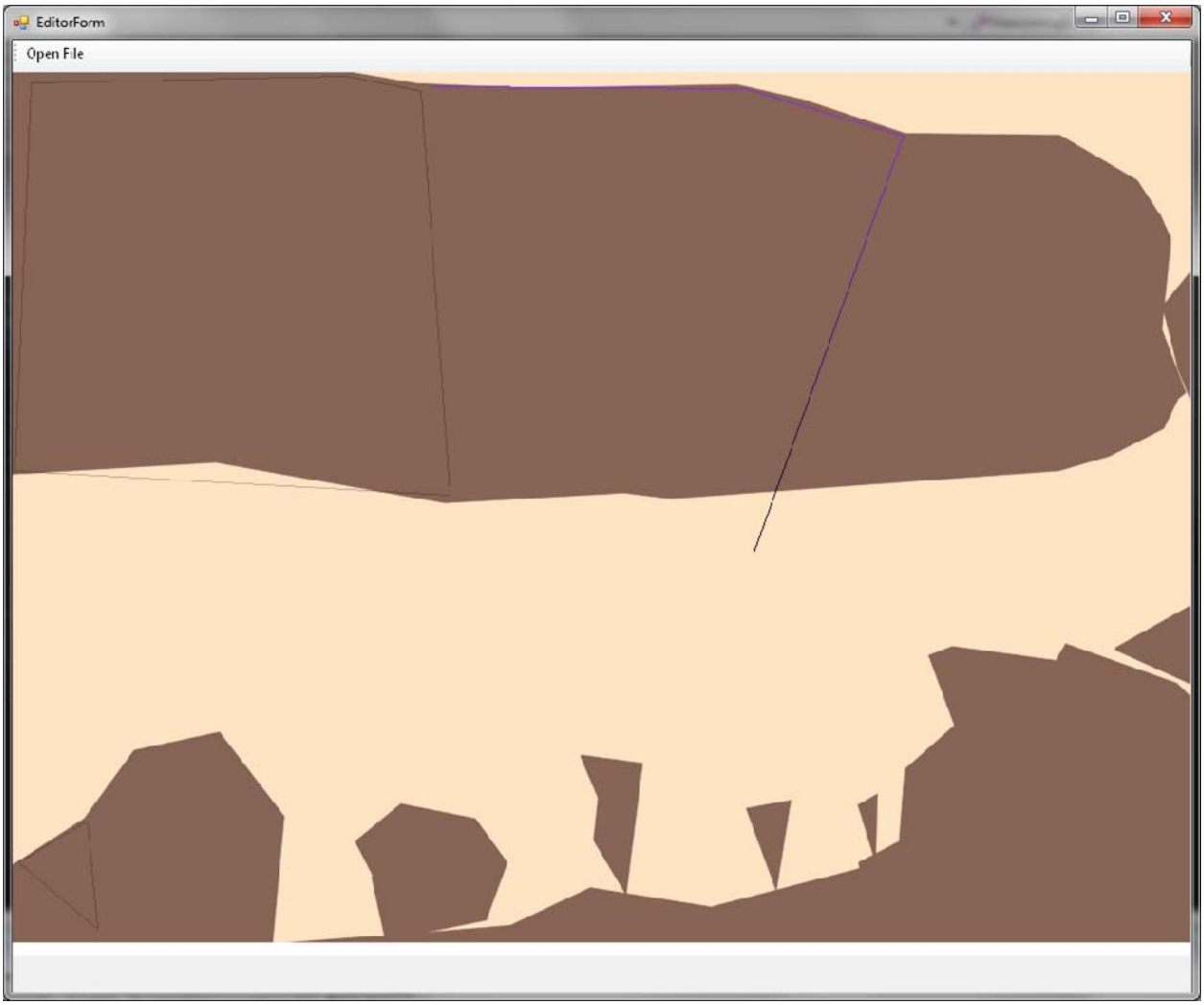


Figure 10 – Rough mockup of level 1 within the editor.

The Tech team would import the rough Photoshop mockup into the level editor. Then, using a polygon tool in the level editor, they would draw closed polygons around areas that needed solid ground associated with them. This way they could block out areas like walls, platforms, and floors in an easy, visual style. They then exported these polygons in an image file and sent them to the Art team for a paint over.

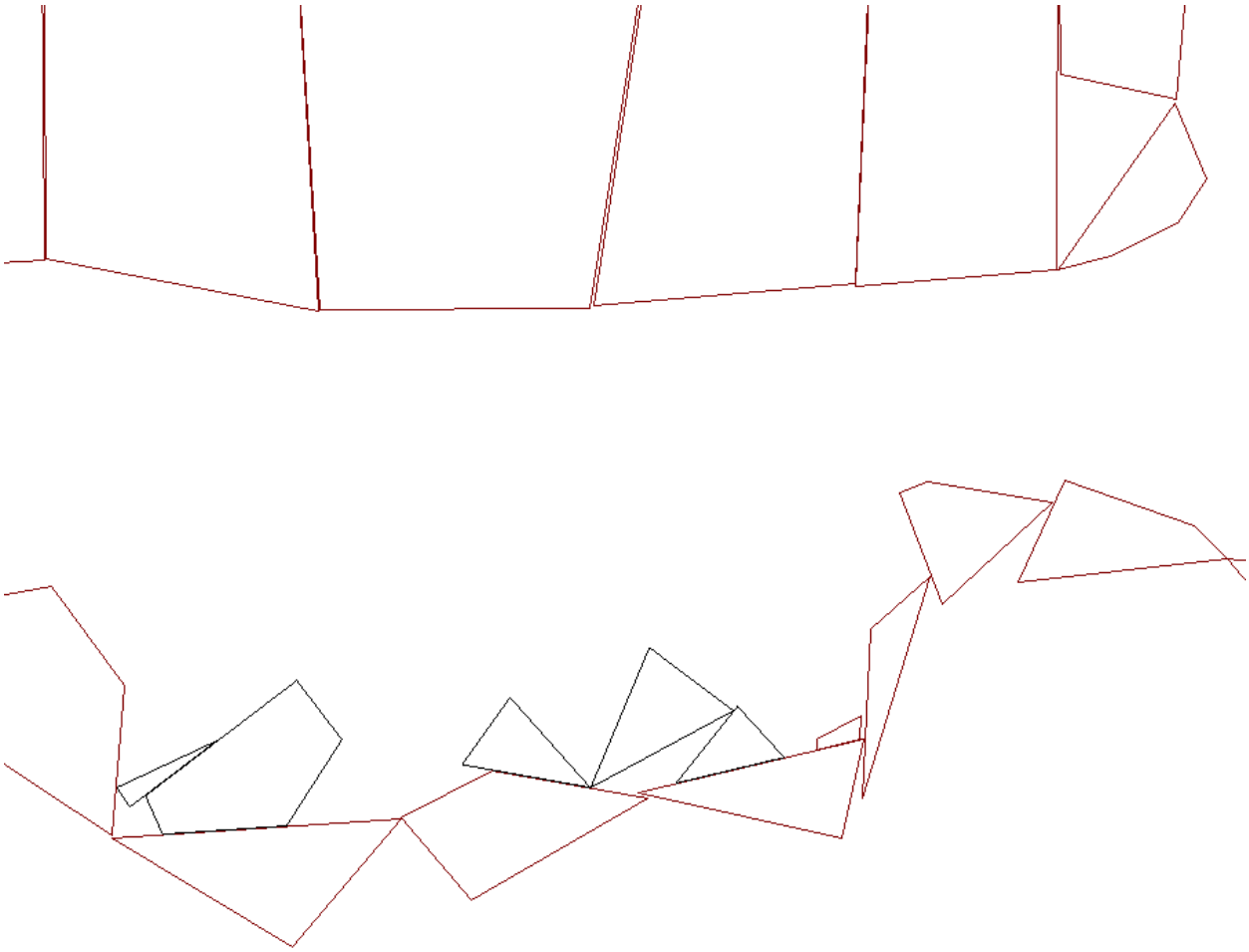


Figure 11 – The polygon image file ready to be painted.

Paint Over Polygons in Flash

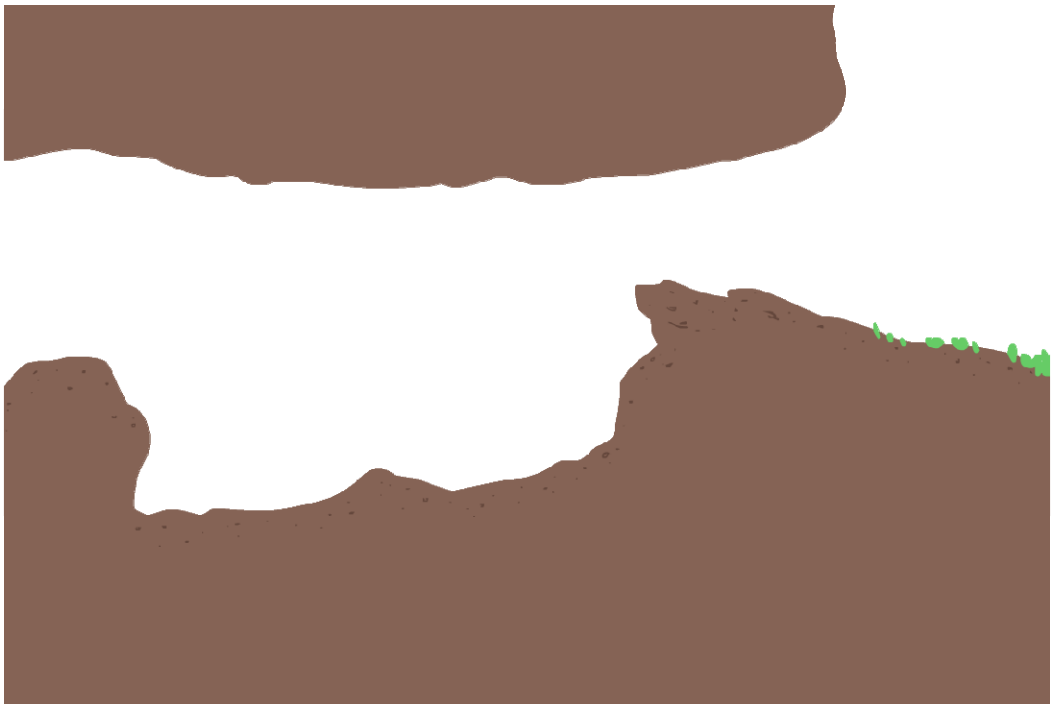


Figure 12 – The ground asset after being painted over in Flash.

The Art team would import the polygons into Flash and would use the vector painting capabilities to paint over the polygons. The artists had to be careful to make sure that the imported polygons maintained their original size. Due to the large image files we used for each level, this did not always happen, thus cross-checking the original dimensions became important.

The Art team needed to add aesthetic value to the blocky environment presented by the polygons, but also had to be sure that the art would cover every polygon to keep the player from bumping into an invisible wall or "hovering" over the ground.

Figure 13 – Left: Gideon is right on top of the ground, Right: Gideon is sunken into the ground by a few pixels.

The Art team also couldn't go too far over the polygon line or the player would fall into the ground too far. Gideon stays almost on top of the "ground" of each level, with the exception being the museum level where he sinks in quite a bit. This was done to create depth and imply that Gideon was in a large room.

Once the polygons for the level were painted over, the Art team would export them in Flash as a PNG file with "transparency" and "minimum image size" options checked to ensure that the assets would fit nicely into the editor.

Create Additional Assets

Special Assets for Each Level

Figure 14 – Dynamic assets in level 1.

While the ground and walls of a level were usually one large static asset, the puzzles in each level often required dynamic assets, usually comprised of things that could easily break down or collapse

into smaller platforms. For these assets, there were two ways of determining what they would look like. Tech would draw black polygons over dynamic assets so that Art would know that these weren't part of the large static asset. From there, Art could go back to the polygons to draw over them for the static assets. This worked well for nondescript dynamic objects like rocks.

Some dynamic objects were more specific. Tech would tell Art the general size they were looking for in an asset and Art would usually create the asset and immediately show it to Tech for review. Tech would recommend any changes in size or shape and then Art would go from there. After a few iterations of this, Art would send the assets over to Tech for implementation. Since this process was generally done side-by-side in the IMGD lab, this went rather quickly and was usually quite successful. If for whatever reason Art couldn't meet with Tech or vice versa, this process became a bit harder and sometimes either completed assets would not make it into an iteration or the assets would not be completed on schedule.

General Assets for the Game

Figure 15 – Gideon's Walk Cycle, an asset used throughout the game.

Many assets for the game were common to all the levels, such as the assets for Gideon and his walk cycle. These assets were created with our lined, colorful, flat theme in mind, usually with some type of reference associated with it. The title screen, for example, draws inspiration from the map of the meat on a pig. The instruction screen and much of the UI of the game draws from the style we developed for the *Chronopigger* website.

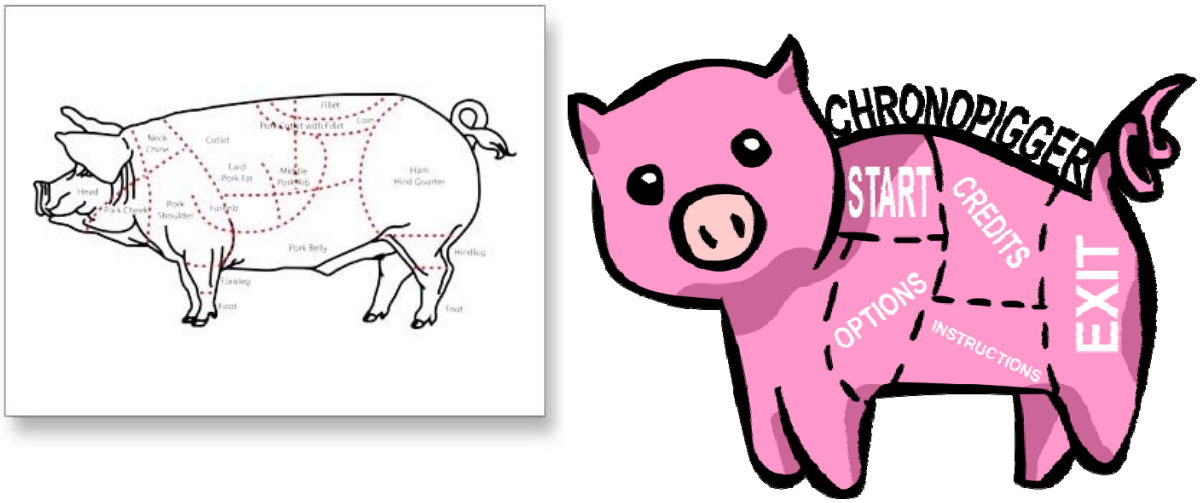


Figure 16 – The humor of the title screen.

Creating Music for the Game

We got in contact with Jared Cowing, a fellow Worcester student who offered free custom music for games. We worked with Jared over email to establish the mood and melody that we wanted for the overall game and for each individual level. Sometimes we would request tracks for specific emotions, such as a "tense" track or the "windy" track used for the fans in level 2.

Send to Tech for Implementation in Game

Figure 17 – A nearly complete level 1, displaying polygons and finished assets.

As the Art team produced each batch of assets, they sent the assets to the Tech team by committing them to the Subversion repository or by emailing them. Email became the most popular method of sending assets since sometimes Art was unsure of which assets Tech would want to implement at the time.

Post-Mortem

Many game developers write post-mortems for their games to analyze the process and learn from the experience. These are generally presented to other developers and the public in papers, articles or talks with the hope of imparting knowledge to the community. Post-mortems are structured as pros (what went right), cons (what went wrong), and analysis (what we learned). Much of the value of our project was in the experience we gained throughout its development.

What Went Wrong

Recording physics causes headaches

The recording system turned out to be a more difficult problem than anticipated. Our trouble was split into three categories: design, implementation bugs, and miscommunication/misunderstanding. The design was flawed in that physics events were treated like any other event: they exist at some arbitrary time and execute when the game time passes their stamped time. It would have been much better to have a consistent physics update interval and to store the events based on knowledge of the interval. We had several bugs where events would be skipped or executed multiple times because of this fact, and it was not trivial to separate physics events from the other events. As far as implementation goes, the editor and the game both had some bugs related to the recordings. The editor would sometimes place events out of order or before the beginning of the timeline, causing the saved recordings to have some incorrect data. The game had inconsistent execution of recorded events, resulting in saved recordings sometimes starting with broken polygons or missing events.

Not enough focus on game design

Our game design was flawed in terms of the types of potential puzzles we had. Looking at the game in abstract, it seemed like we would have a good number of different puzzles to design into the levels. When we started implementing the game mechanics and level editor, it became apparent that we were going to need some more mechanics for it to remain fun throughout the game. We came up

with a few ideas and settled on the Time Glue and Time Loop items which the players can use to manipulate the world. This gave us a number of new possibilities, but it seemed to be too little too late. It would have been a good idea to put more emphasis on finishing and polishing a single level earlier, as opposed to trying to make assets and write features for multiple levels. This would have let us find gameplay problems earlier and end up with at least one really solid and fun level.

Missed deadlines and poor communication

Throughout the project, we had trouble figuring out milestones and communicating between the tech and art teams. Asset lists weren't maintained, design docs weren't updated, and there was an overall lack of process when it came to the weekly iterations. Having a more formalized weekly and long-term plan would have helped the organization of the project immensely. As for communication and scheduling, we made the poor decision of having the project team contribute different amounts of work over the 3 terms of the project. One team member came in a term late, which left him out of the loop for the decisions made during the formative stage of the project. Additionally, the day-to-day classes and commitments that each team member had made it difficult to find weekly meeting times to work on the project as a team.

Artists with different styles

The artists had two very different styles that remained mostly separate all the way up until the final hours of game creation. Each was used to working in different drawing software, and when the cartoony style was agreed upon it became evident that Flash was the program that most of the game would be made in. Thus the artist with less Flash experience had to learn the software while simultaneously producing usable assets.

Pipeline bottleneck

Due to the designer problem, the art pipeline suffered significant setbacks as the art team waited for Tech to produce polygons before they could start any work on the next level, since level

specific assets couldn't be made until the game "room" was decided upon. Unfortunately, polygons usually got sent out late in an iteration, giving Art very little time to finish them with any sort of polish. Had there been a dedicated level designer, these game rooms and polygons would have been completed much earlier in each iteration, giving the art team more time to spend generating and polishing assets.

What Went Right

Not reinventing how wheels spin

Using a third party physics library was a great decision. Implementing physics takes a long time and we would likely have gotten it wrong. None of the team members have a strong enough math or physics background to do it right the first time.

Making our own engine

Our early technical decisions were well-founded and proved to be good ideas. Using XNA meant that we could develop an engine with the features we needed from the ground up. We also didn't have to worry about performance until late in the process; if we had opted for Flash then performance might have been a significant problem.

Making our editor early

Creating a level editor as an initial step was beneficial, though we didn't use it enough at the beginning. It showed us where our gameplay design was flawed and prompted us to try to fix some of the problems early. Because we were able to test level ideas quickly, we discovered that we were having trouble coming up with puzzle ideas. Near the end of the project, the fact that we had made the editor early meant that we had time to fix some of the bugs instead of worrying about implementing features at the last minute.

Scalable art

Using Flash allowed the artists to produce vector assets. This choice made the majority of the game assets infinitely scalable, so whenever there was a size issue it was usually quite easy to fix. Flash in general allowed for many shortcuts to be taken, which helped keep the Art pipeline running quickly. While the pipeline was quick, it was not without its limitations and setbacks.

Marketing

Figure 18 – The home page of the *Chronopigger* website

While not directly part of the game itself, a small marketing campaign was launched during the project. In order to increase the likelihood of being recognized at the 2011 IGF, we began creating a web presence for *Chronopigger*. This presence consisted of a website designed by the team as well as a Facebook group. Both the website and Facebook group received an overwhelmingly positive response despite the game not being complete at the time both were created. The positive feedback we received

provided a boost to team morale and provided increased motivation to polish the game. Finally, the style of the website ended up influencing parts of the game, particularly the instructions, credits, and title screen. The style made excellent use of negative space and provided a punchy, modern look to capture the interest of internet audiences.

Outsourcing music

Coordinating with someone outside the group was an excellent decision overall. While we were initially hesitant that working with a third party would not yield useful results, Jared Cowing pleasantly surprised us by delivering quality music on a consistent schedule. His compositions gave the game a unique audio style that it would have otherwise lacked. For sound effects, the team used Creative Commons assets from The Freesound Project (The Freesound Project, 2011). Rather than spending time generating our own sound assets, we made full use of the quality audio available to us, freeing up valuable development time.

What We Learned

Teams need producers

We had a lot of trouble when it came to organization, communication, and scheduling. For these reasons it would have been very useful to have someone in a producer role for the project. It probably would have been sufficient to assign the role to one of the team members, as opposed to acquiring a sixth team member. While this person would likely be expected to accomplish less of their main job, their production work would increase the overall quality of the team's work as a whole. Having a producer could have provided enhanced art/tech communication, personal accountability for each team member, and planned milestones. The producer also could have ensured that the team maintained the same vision throughout the project by keeping design document updated with the official decisions regarding the game.

Artists and programmers aren't designers

For a game that relied so heavily on having interesting puzzles that showcased the game mechanics, we should have had someone in a dedicated design role. While the role may have switched hands from week to week, it was difficult for members of the tech team to manage both level design responsibilities while still developing the technology behind the game. Rather, we should have relieved one member of the tech team of their tech responsibilities each week and had them focus solely on making sure levels, puzzles, and gameplay situations were challenging and fun. Repeated brainstorming, prototyping, and testing could have validated level and mechanic design early in the process so that there would be plenty of time for iteration.

Quantity over quality when brainstorming

When it came to levels, we made the mistake of expecting the first few levels we designed to remain in the final version. Rather than coming up with and implementing a wide variety of levels, we spent most of our time brainstorming levels on a whiteboard or iterating on the few levels we'd put in the editor. We had assumed that, given our mechanics, it would be easy to generate levels with interesting gameplay. However, once we started designing levels using the mechanics, we started to realize how difficult it would be. Additionally, the editor may have been too technical. With sparse documentation after the refactoring, it was difficult for the art team to contribute to level design. As such, we eliminated a lot of possible experimentation by limiting level creation to only the tech team.

Code walks should be often and early

The code walk provided valuable insight into various weaknesses in our code or general areas that the code could be improved on. However, the code walk happened too late in our development cycle for us to have time to implement the changes. Additionally, by the time that the code walk occurred much of the engine was in the same state it would remain for the rest of the project. Thus while refactoring may have yielded an increase in extensibility, the new avenues for extension would go

unused. As a takeaway, we wish we'd prepared for the code walk earlier so that the valuable information we took from it could be properly acted on.

Conclusion

Despite our efforts, *Chronopigger* is not ready for submission to any major competitions. It's sorely lacking in game design and still needs to undergo polish passes. However, we had anticipated on only having a beta version of the game by this point. With eight months remaining until the IGF submission deadline, the team still has plenty of time to refine the game.

Bibliography

15-Puzzle. (2006, August 16). Retrieved Febrauary 28, 2011, from Wikipedia: http://en.wikipedia.org/wiki/File:15-puzzle.svg

Continuity. (2010). Retrieved February 28, 2011, from Gamasutra: http://www.gamasutra.com/db_area/images/igf/Continuity/screenshot.jpg

IGF Student Showcase 2010. (2010). Retrieved February 28, 2011, from Independent Games Festival 2010: http://www.igf.com/02studentfinalists.html

Paper Cakes Screen 2. (2010). Retrieved February 28, 2011, from Paper Cakes: http://papercakes.nl/images/Scrn_PC_002.png

Paper Cakes Screen 3. (2010). Retrieved February 28, 2011, from Paper Cakes: http://papercakes.nl/images/Scrn_PC_003.png

The Freesound Project. (2011). Retrieved February 28, 2011, from The Freesound Project: www.freesound.org

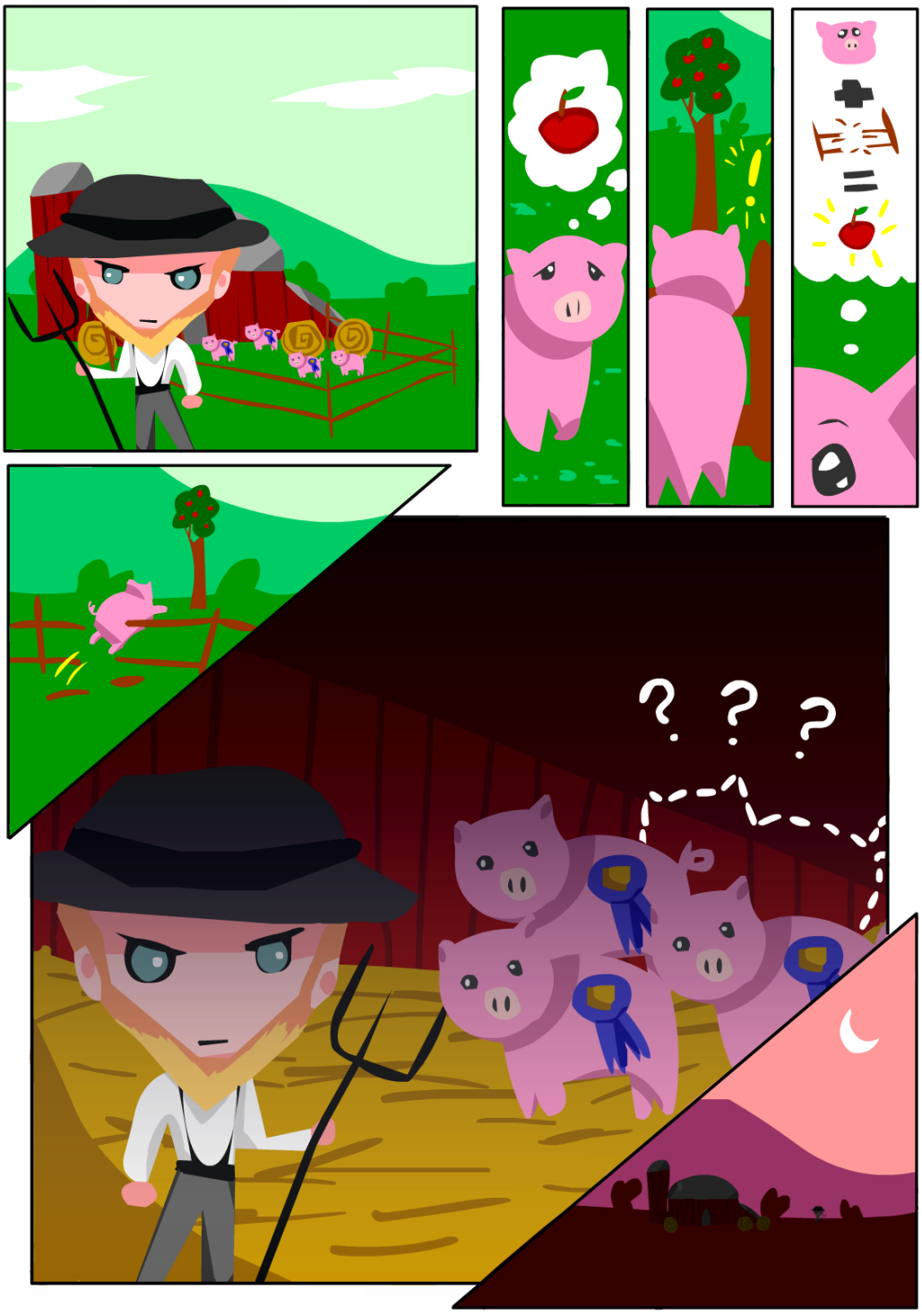
The Story of The Hex Sign. (2010, February 21). Retrieved February 28, 2011, from Amish Country News: http://www.amishnews.com/featurearticles/Storyofhexsigns.htm

Witness. Weir, P. (Director). (1985). [Motion Picture].

Appendix A: Art Assets

Below are samples of art assets that went into *Chronopigger*.


Nice.

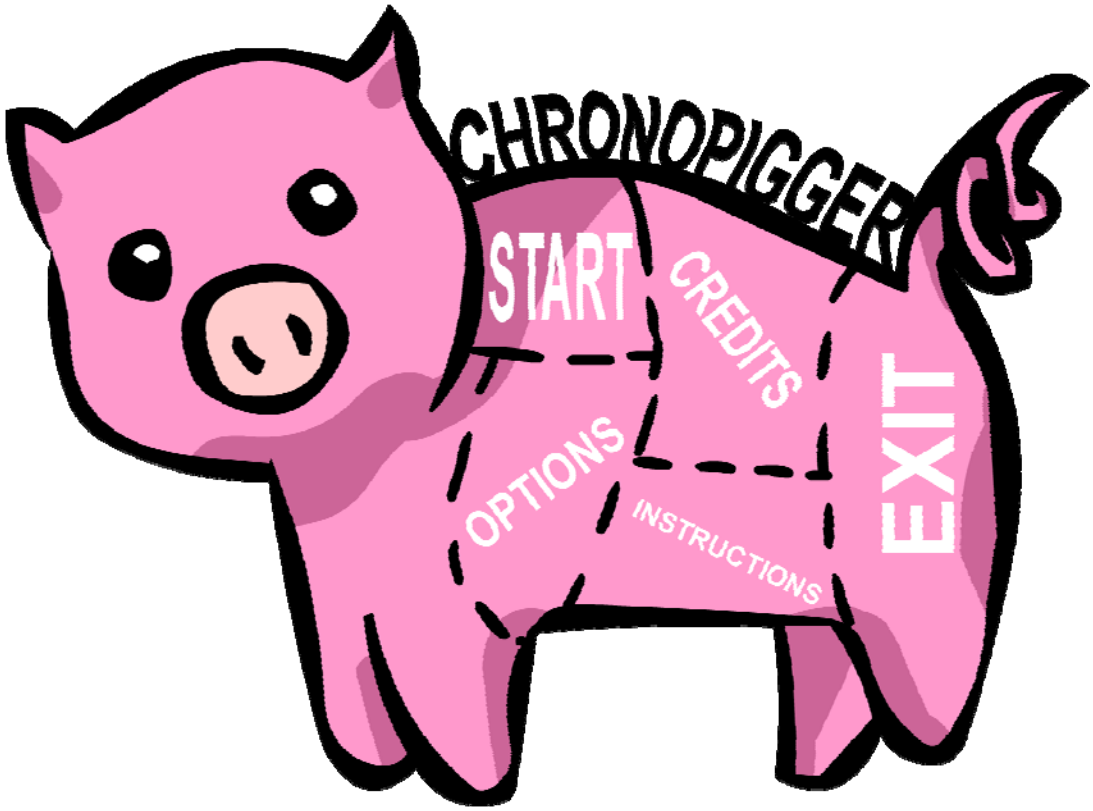


Above are assorted comics that were used as cutscenes in *Chronopigger* to explain the story.

Gideon's walk cycle.

Amos.

Various gears used on the hub screen.



The *Chronopigger* title screen.