MQP MBJ 1602:
MINIMALIST STORYGAME

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

Jeffrey Bardon
Shane Stenson

April 28, 2016

Advisor:
Brian Moriarty, IMGD

Abstract

This report describes the design and development of two games inspired by literary sources: Thomas Mann's *Death In Venice* (1912) and Ambrose Bierce's *The Devil's Dictionary* (1906). The game based on Mann's novella posed significant creative and technical challenges, only some of which we were able to address satisfactorily. Halfway through the project, we switched to *Dictionary*, which afforded greater flexibility in scope, and successfully assembled a collection of six mini-games, each based on one of the book's cynical definitions: Painting, Litigation, Longevity, Academy, Coward and Bore.

Acknowledgments

First and foremost, we'd like to thank our advisor, Professor Brian Moriarty, for allowing us to make the progress on *PermaDeath In Venice* that we did, his invaluable guidance and input regarding both of our projects, and his understanding during our transitionary period. Thanks also to Drew Tisdelle for his assistance with configuring source control while we were trying to get the project started, and also to the people who played and tested our game. Finally, much thanks to Thomas Mann and Ambrose Bierce, for authoring rich source material to draw from for our games.

Contents

1. Project conception

The Minimalist Story Game MQP was conceived with the intention of creating a game using Perlenspiel, an abstract micro-engine created by Brian Moriarty (our project advisor) in 2009 for use in his game design classes at WPI. Inspired by Herman Hesse's 1943 novel *Das Glasperlenspiel (The Glass Bead Game),* Perlenspiel "challenges designers to work in a raster of jumbo-sized pixels or 'beads' no larger than 32 x 32, the dimensions of a standard Windows icon." (Moriarty)

Our goal was to produce an abstract adaptation of Thomas Mann's *Der Tod in Venedig (Death In Venice),* a novella published in 1912 by Thomas Mann, winner of the Nobel Prize for acclaimed novels such as *Buddenbrooks, The Magic Mountain* and *Doctor Faustus*. The story follows an accomplished German writer named Gustav Von Aschenbach, who, in a fit of wanderlust, finds himself drawn from his native land to the city of Venice. (Mann) There he first encounters a Polish boy named Tadzio, with whom Von Aschenbach grows increasingly infatuated to the point of obsession.

As his days become consumed with following Tadzio and his family around Venice, Von Aschenbach becomes aware of an epidemic of cholera sweeping the city. Despite this knowledge, and the associated danger, Von Aschenbach's reason gives way to passion; he chooses to remain in the city and continue his fervent pursuit. Eventually Von Aschenbach falls victim to the plague, dying on a beach while watching Tadzio gaze out across the sea.

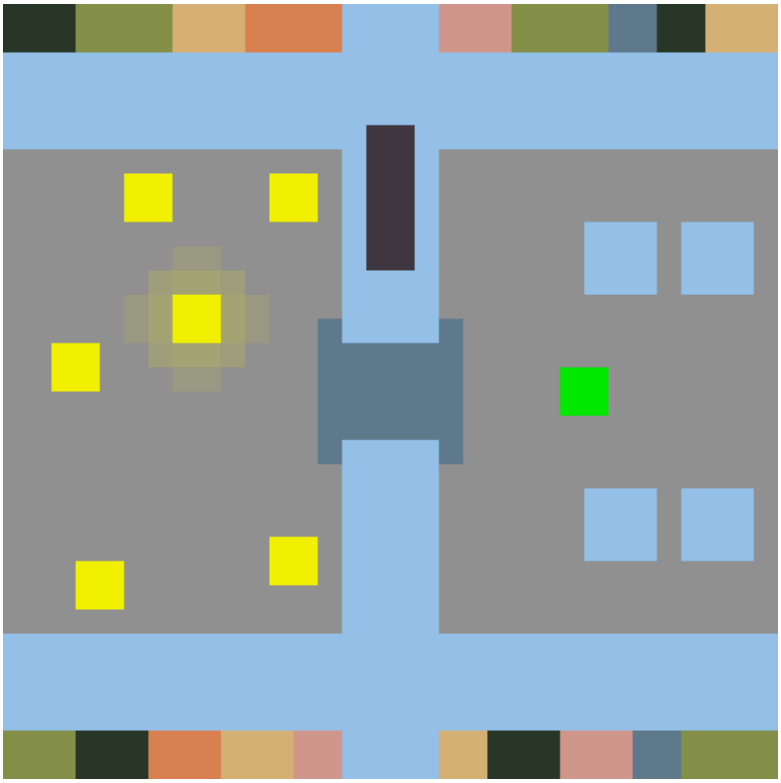

Figure 1. Scene from the original 2012 version of *PermaDeath in Venice*, showing Aschenbach's first encounter with Tadzio.

Moriarty had previously created a simple adaptation of Mann's story for use in a 2012 Game Developer's Conference lecture demonstrating the Perlenspiel engine (Figure 1). His game, called *PermaDeath in Venice (PDV)*, was subsequently reworked twice, but Moriarty was dissatisfied with the result and never released it publicly. In 2015, he offered *PDV* as a prospective IMGD MQP in hopes of finally achieving a game worthy of its source material.

The original *PDV* game ended with Aschenbach dying of cholera (simulated by slowing his movement speed) while approaching a motionless Tadzio on the beach. Upon reaching Tadzio, Aschenbach's color changed from red to black, slowly expanding to fill the entire screen, leaving only Tadzio's golden halo. Moriarty felt this ending failed to capture the spirit of the original, and also presented a serious mechanical issue: What if the player chose *not* to approach Tadzio? We therefore started our project by attempting to design a better way to handle the ending.

2. Design issues

We went through many ideas, including having the player control Tadzio for the finale. However, this inversion of the game mechanics risked confusing the player, and did not address the problem of an uncooperative player. We finally settled on "killing" Aschenbach just *before* he reaches Tadzio, and then passing control of Tadzio to the player, who could then choose to investigate Aschenbach or walk away, ending the game. This solution seemed to offer a choice more aligned with the spirit of the story.

2.1. Tadzio's behaviors

Among many things, *Death In Venice* is a story about obsession. One of our primary goals with *PDV* was to evoke in players a similar sense of obsession by tasking them with the pursuit of Tadzio through Venice. We hoped to accomplish this by careful design of the geography, and by fine-tuning the algorithms used to model Tadzio's response to the player's pursuit. These responses fell into three categories: Indifference, Avoidance, and Flirtation, each varying in complexity over the course of the game.

2.1.1. Indifference

While operating under Indifference, Tadzio would meander around Venice in an apparently aimless fashion. His "awareness" of Aschenbach would encompass a wide radius, causing him to move vaguely away if Aschenbach came anywhere close, and stop/wander if the approach was abandoned. Our intention with this "loose" mechanic was to convey the idea that Tadzio had yet to take full notice of his pursuer.

2.1.2. Avoidance

The Avoidance behavior was to be the most direct and simplistic of Tadzio's behaviors. In this mode, Tadzio would more obviously avoid Von Aschenbach, typically staying close to the edge of the screen, with the player only getting brief glimpses before Tadzio would slip away. The maps where Avoidance would take effect were designed in a rather maze-like manner, consisting of narrow paths with many branches, and covered areas to obscure the player's view of Tadzio.

2.1.3. Flirtation

The final behavior, Flirtation, was the most complex. Near the end of the novella, Aschenbach sees (or believes that he sees) Tadzio beginning to offer a few glimmers of reciprocation. We hoped to convey this by allowing Aschenbach to get much closer to Tadzio, often within one or two spaces, before the boy would shy away. If Tadzio moved off-screen, he would soon return to wait for his pursuer to catch up. This behavior would be staged amid the canals of Venice, with Tadzio often mirroring the movements of Aschenbach from the opposite side of the water.

2.2. City design

Our map design for the city of Venice went through many stages. We first needed to decide whether the map was going to be procedurally generated from a collection of interlocking modules, or purposefully designed as a whole. At this early stage, we had yet to determine how the "permadeath" of our title was going be implemented, did not know how the passage of time would be modeled, and had not identified which behaviors Tadzio ought to exhibit. With so many unknowns in mind, we decided to rely on hand-crafted maps.

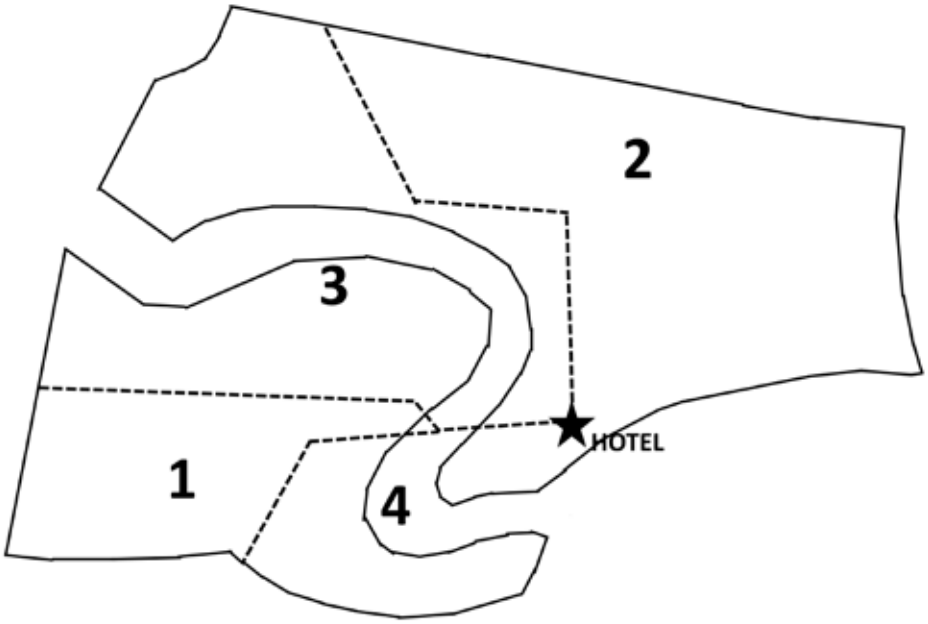

Figure 2. Original map design.

Initially, the city of Venice was implemented as a single, sprawling map with well-defined districts, the intention being that Tadzio would (beginning from the starting location of the hotel) lead Aschenbach through different sections of the map on different days, designed around Tadzio's varying behavior, and returning to the hotel at the end of each day (Figure 2).

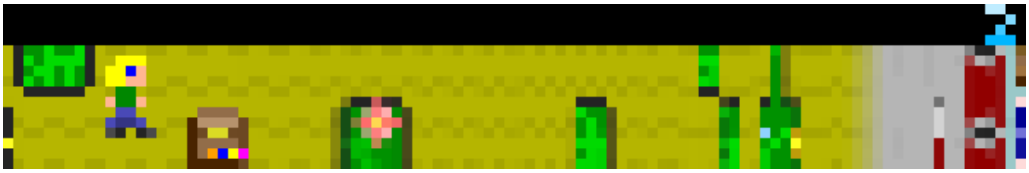

Figure 3. Screen shot of Jason Rohrer's *Passage* (2007).

Inspired by the game *Passage* by Jason Rohrer (Figure 3), we eventually developed a new plan in which each episode is separated into two sections. The first section is a time-based path where the player can only scroll the map to the right but not left, representing the irreversible passage of time. The second section of the map became spatially-rooted paths that branched off the central, time-based path (Figure 4). In these sections, the player is free to move in all directions, an important ability, considering this is the section in which the pursuit of Tadzio actually takes place.

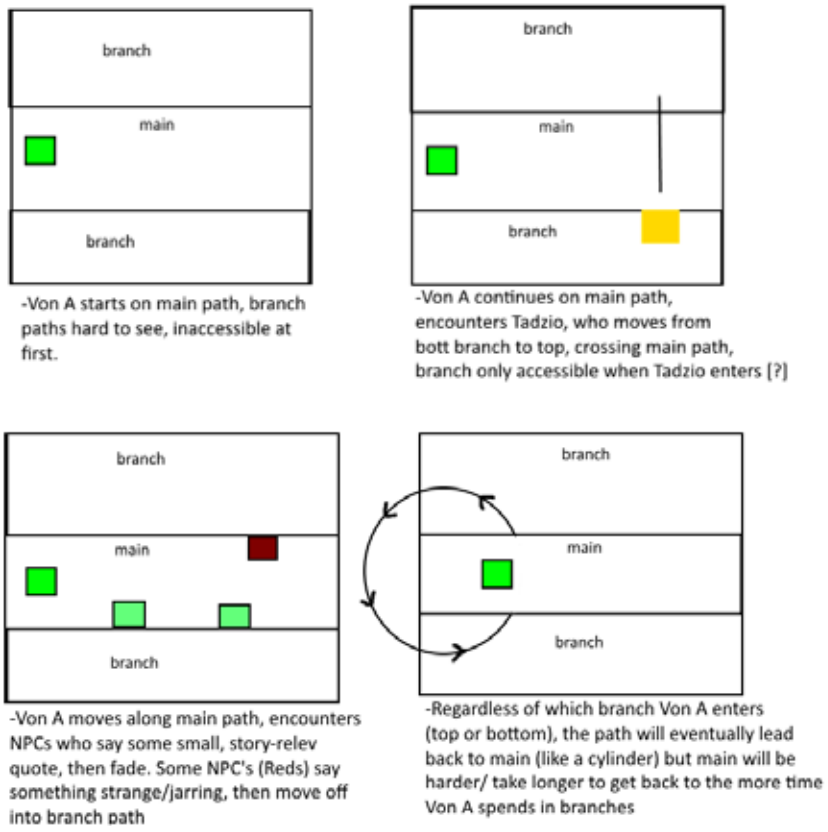

Figure 4. Final map structure.

The intention behind this design was to highlight the contrast between the rigid structure of Von Aschenbach's life as conveyed in the novella, and the deleterious effect his mounting obsession has on this structure. We also switched to individual maps for each episode of the game, with each episode spanning a "day" in Venice or a dream at night.

2.3. Permadeath

"Permadeath" is the industry term for games which allow a player to lose without recourse. We knew our game would incorporate permadeath from the beginning, but it took some time to decide on an implementation.

Originally, we assumed that the permadeath of the title would refer to the *player*; that is, someone could play the game once, and only once (unless they tried to play again on a different machine, or utilized some other workaround). However, we eventually settled on a version of permadeath focused on Tadzio. The plan was that players would be able to use their actions in the game as a sort of "vote." Those who doggedly followed Tadzio around the side paths would effectively be voting to keep Tadzio in the game, while those who chose to ignore Tadzio and keep to the main, linear paths would be voting for his removal. If enough players ignored Tadzio, the character would be permanently removed from future playthroughs.

This odd mechanic would have been implemented by having *PDV* write data to an external server as players played the game. Choices at key moments would be recorded and tabulated to determine whether or not Tadzio would appear. We quickly realized that an absolute threshold would inevitably be reached, even if by accident, while a percentage-based threshold would run the risk of triggering immediately. To get around this, we considered a combination of these two methods, and only remove Tadzio if enough people had played the game *and* a certain percentage of players had ignored Tadzio. The final algorithm was never determined.

2.4. Episodic structure

PermaDeath In Venice was designed with an episodic structure, broken into five main episodes. It began with Aschenbach already in Venice, and ended with his death on the beach, the episodes in between representing Aschenbach's actual time spent in Venice, and his pursuit of Tadzio.

Our design also included four dream sequences, one between each of the five main episodes. Some were flashbacks explaining the decisions and actions that led to Aschenbach's arrival in Venice. Others were more abstract, dealing with Aschenbach's state of mind and feelings toward Tadzio.

Figure 5. Map of Episode 1.

2.4.1. Episode 1

Tadzio Behavior: Indifference

The game's shortest and most linear episode, Episode 1 begins with Aschenbach in his hotel room and leads into the city itself, along the time-based path (Figure 5). The spatial side paths factor into this episode minimally, only appearing near the end, when the player enters the hotel beach. This is where the player first encounters Tadzio, who exits one of the private bungalows and begins to move up the length of the beach, into the side path. The player follows Tadzio on a brief walk along the beach before returning to the hotel at the end of the episode.

2.4.2. Dream 1

This brief sequence was intended to introduce the idea of the interstitial dreams. It begins with the player on a gondola, in transit from the steamer Aschenbach took to Venice. After a brief trip across the harbor the player reaches the hotel dock, exits the boat, and enters the hotel, ending the dream.

2.4.3. Episode 2

Tadzio behavior: Indifference, changing to Avoidance

Here the side paths become a prevalent, built here mainly of wide walkways and open courtyards across the city. Near the end of the level, these begin to give way to narrower alleyways, becoming more maze-like and claustrophobic as Tadzio begins trying to actively avoid Aschenbach and increase the difficulty of his pursuit.

2.4.4. Dream 2

This dream takes place in Aschenbach's native Germany, leaving the player to navigate its gray, washed-out, seemingly barren streets. Eventually, they would come to a chapel, on the steps of which rests a red square. As the player moves closer to the square, the screen begins to fade, transitioning to Aschenbach on a steamer, boarding the gondola that he rode in Dream 1. This dream emulates the beginning of the novella, in which Aschenbach, walking around his home town, notices a skeletal red-haired foreigner (represented by the red square) who evokes in him visions of exotic, far-flung locales, inspiring him to take the trip that brings him to Venice.

Characters incorporating the color red are a recurring element throughout Mann's novella. We have interpreted them as representative of the passion and disorder that begins encroaching on Aschenbach's disciplined lifestyle. We hoped to incorporate this element of the story by having a variety of red non-player characters (NPCs) throughout the game, particularly in Dreams 2 and 4, but never decided the details of how they would be used.

2.4.5. Episode 3

Tadzio behavior: Avoidance

The game's most difficult episode, at least in terms of pursuing Tadzio. The map for this episode was the most labyrinthine, built from narrow alleys with numerous branching paths, and stretches of covered street, obscuring Tadzio, Aschenbach or both from the player's view.

2.4.6. Dream 3

Dream 3 didn't get completely fleshed out, design-wise, but would've likely involved the player running through the map from Episode 1, though this time with Tadzio appearing in numerous sections, often in ways he shouldn't be able too, the manner of which would've been influenced by whether the player regularly pursued Tadzio through the side paths, or opted to stay primarily along the main path.

2.4.7. Episode 4

Tadzio behavior: Flirtation

Somewhat similar to Episode 2, with a layout consisting mainly of wider, open areas where Tadzio would often linger, allowing the player to get very close before moving away again, sometimes even re-entering an area he'd left to wait for the player. In addition, Episode 4 would highlight the city's canals, with the player walking along their banks and Tadzio on the opposite side, just out of reach, but almost mirroring the players movement, moving away from the canal only when the player crossed a bridge or took a gondola to reach the other side.

As previously noted, Aschenbach eventually becomes infected with the cholera plaguing the city, succumbing to the disease in the book's final moments. Though we planned to incorporate the moment when Aschenbach becomes infected in our game, the implementation details were never decided, but it would have happened either at the end of Episode 4 or the beginning of Episode 5.

2.4.8. Dream 4

This was the most complex dream in our design, and the only one actually drawn from the novella, though staged quite differently. It begins with the player in a maze, similar in appearance to earlier episodes of the game, but fractured, surreal, and tinted in shades of red. A red NPC, similar to the one seen in Dream 2, begins to chase the player through the maze. More and more red NPCs appear the longer Aschenbach manages to evade them. During this pursuit, Tadzio is frequently visible, unmoving and inaccessible in walled off sections, watching the player's flight and eventual capture.

Upon the player's inevitable capture, the action would abruptly cut to a small, white room, occupied only by the Aschenbach and Tadzio. No matter where the player clicks in this area, Aschenbach moves towards Tadzio, but never more than one space at a time. As Aschenbach approaches, Tadzio moves towards the opposite wall, the screen getting smaller and darker with each step. Soon Tadzio would be against a wall, unable to move as Aschenbach continues to approach. As they meet, the screen cuts to black, ending the dream.

2.4.9. Episode 5

Tadzio behavior: Flirtation

This episode was never fully developed, partially due to hesitation on our parts in trying to convey its importance. For purposes of dramatic symmetry, Episode 5 would have likely been similar to Episode 1, and ending on the same hotel beach, with Tadzio standing at the water's edge. Upon approaching, Aschenbach would succumb to his illness, stopping dead in the sand. At this point, the player would briefly take control of Tadzio, and could decide either to approach the fallen figure of Aschenbach, or turn to the water and wade out further. The game ends with either decision.

3. Technical issues

3.1. Source control

One would think that configuring source control would be a trivial task, but our team experienced problems with every step of the process. The first repository we created ended up being inaccessible, and the second one wasn't much better. For most of A Term, we found ourselves sending files back and forth over our Slack channel, which proved to be an invaluable resource during the project. Slack's always-available line of instant communication, together with its simple file transfer and archiving facilities, literally saved the project at a couple points.

At the beginning of B term, we finally managed to get a BitBucket repository up and running with SourceTree. However, SourceTree eventually glitched and deleted every project file in the repository. We wearily elected to return to direct file transfers for C and D Terms, rather than go through the risk of attempting another repository.

Figure 6. Visual layer of Episode 1, which controls the background actually seen by the player.

Figure 7. Pathfinding layer of Episode 1, which controls where the player and NPCs can (white) and cannot (black) move.

3.2. Image loading

As *PDV* was to be built with Perlenspiel, the maps used would have to be either encoded into the source files as arrays, or dynamically read from image files. For ease of testing and iteration, we implemented image loading. Perlenspiel's built-in facility for A* pathfinding made this aspect of creating the maps straightforward. Transitioning from episodes to dreams and back again was accomplished simply by loading in a new set of maps.

We had devised a means of encoding all relevant information (geography, walk paths, player and NPC locations, triggers, etc.) within different layers of an image file (Figures 6 and 7). However, the farther into the implementation we delved, the more layers we ended up needing. Eventually each map required over a dozen layers, some of which (such as the triggers) requiring specific RGB values. Debugging these proved troublesome, as depending on the image editing software that saved the image, some RGB data could be lost.

Our first tool choice, paint.net, uses compression algorithms which clamp the original 0-255 pixel data range into only 100 values (changing both 1s and 3s to 2s, for example). This issue set us back approximately a day in the implementation, as there were no bugs in the relevant code, and the original images were saved correctly. We switched to Photoshop to avoid this issue.

4. Project transition

Two terms into *PermaDeath in Venice*, we were still in the early stages of development. The engine wasn't feature complete, though it was close. Though we could have pressed on, we probably would not have had sufficient time to test or implement changes from testing and user feedback. We eventually understood that *PDV* could not finished without sacrificing quality. It was time to find another plan.

After we made the decision to switch projects, we still had to determine exactly what we were going to do. We saw three options, each with their own pros and cons.

The first was to re-implement *PDV* with a different game engine. The upside to this was that we already had a significantly fleshed-out design for the game. But this design was rooted in the peculiar constraints of Perlenspiel. The engine's simple, abstract imagery wouldn't translate well to a more conventional development environment. But switching to "real" graphics would massively increase the artistic load for the game. For this reason, we ruled out sticking with *PDV*.

A second option was making a new game with Perlenspiel. Unfortunately, most of our ideas for alternate games didn't lend themselves to Perlenspiel's ultra-minimalist aesthetic.

Our final option, and the one we ultimately chose, was to start completely fresh with a new game, a new design and a new engine. Even this was not a simple decision, as we had many ideas about what kind of game to create. We elected to continue with a literary inspiration, as there are a multitude of public domain works that would simplify the design process by giving us a predefined theme and structure. We drafted a list of approximately a dozen candidates, tried to come up with game ideas based off of these, and then decided which would be the best design for completing the

project. The final candidates were Richard Connell's *The Most Dangerous Game*, Ambrose Bierce's *The Devil's Dictionary*, Robert Chambers' *The King in Yellow* and Alain & Souvestre's *Fantomas*.

The Most Dangerous Game would have focused on a chase, with mechanics similar to *PDV*, but wasn't selected due to its large scope and complex AI issues. *The King in Yellow* was a potential adventure/horror design, and *Fantomas* was a detective game, neither of which got fleshed out.

We finally settled on Bierce's *Dictionary,* a book of satirical definitions for common English words. Originally published in 1906 as *The Cynic's Word Book*, it is a compilation of material drawn from newspaper columns the author wrote beginning in the 1870s. (Bierce)

Our new game was conceived as a mini-game collection with a cynical spin, similar to Nintendo's *WarioWare* or *Mario Party*, but with mostly loss conditions instead of win conditions. The goal instead would simply be to survive as long as possible to obtain better scores.

This design was highly flexible in scope, a welcome change after our experience with *PDV*. No individual mechanic or game was critical to the overall design, which would allow us to cut games or mechanics that didn't play well or proved too complex to implement.

With a new, modular design in place at the end of B Term, we finally got our MQP back in gear.

5. Project structure

In hope of giving the frame of the game a more complexity than simply presenting the mini-games as a menu of words, we decided to turn the menu into a word search, where, to be played, mini-games need to first be unlocked, accomplished by finding the corresponding word in the word search.

Figure 8. Painting mini-game.

5.1. Painting

PAINTING, n. The art of protecting flat surfaces from the weather and exposing them to the critic.

Painting has the player controlling a canvas that can move around the lower section of the screen, trying to catch falling paint icons to score points, while avoiding the weather falling from the top as well (Figure 8). Meanwhile, critics on either side of the screen offer their "thoughts" on the art in progress, taking the form of speech bubbles that obscure large sections of the screen, hindering the players ability to catch, or avoid the falling objects.

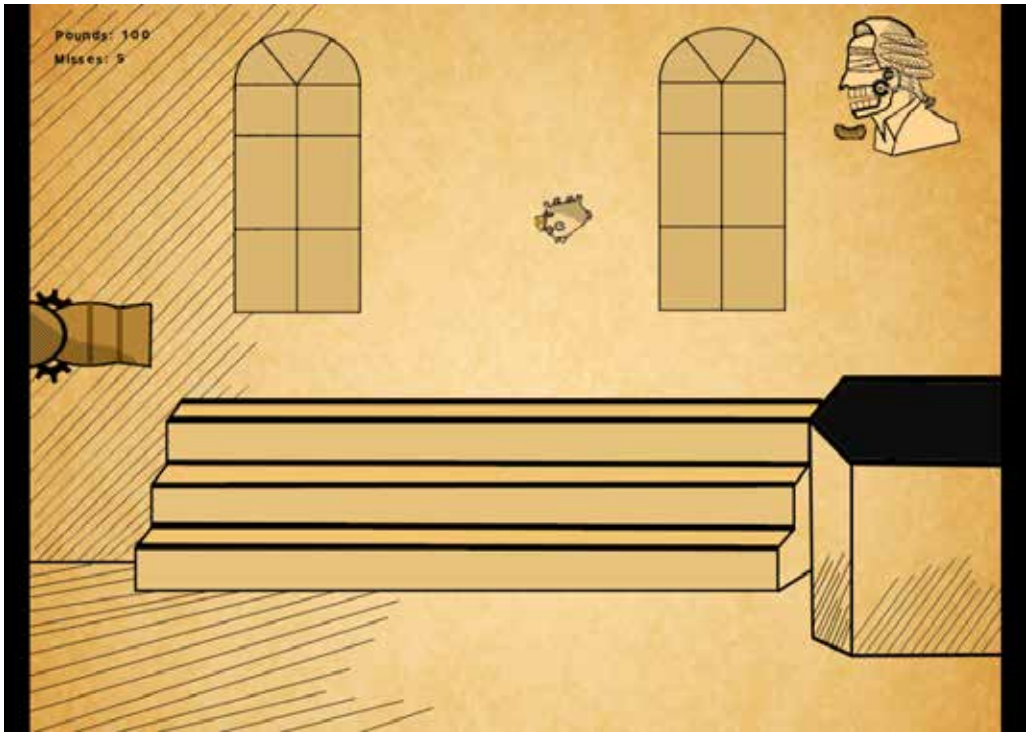

Figure 9. Litigation mini-game.

5.2. Litigation

LITIGATION, n. A machine which you go into as a pig and come out of as a sausage.

Litigation has the player controlling a cannon that's constantly moving vertically along the left edge of the screen, and uses it to fire pigs across the screen, trying to hit the mechanical "judge" machine moving in an identical fashion along the right edge (Figure 9). The judge "eats" the pigs and turns them into sausages, scoring the player points. Missing the judge a certain number of times will lose the game.

Figure 10. Longevity mini-game.

5.3. Longevity

LONGEVITY, n. Uncommon extension of the fear of death.

Longevity is more of a traditional platformer, with the player running rightwards along the level, fleeing the impassable, implacable, and ever-advancing spectre of death at their heels (Figure 10). Their only, fleeting respite being the hourglass pickup scattered across the game, each one pushing death back, a little bit. However, the player will get caught eventually, their final score based on how long they managed to survive.

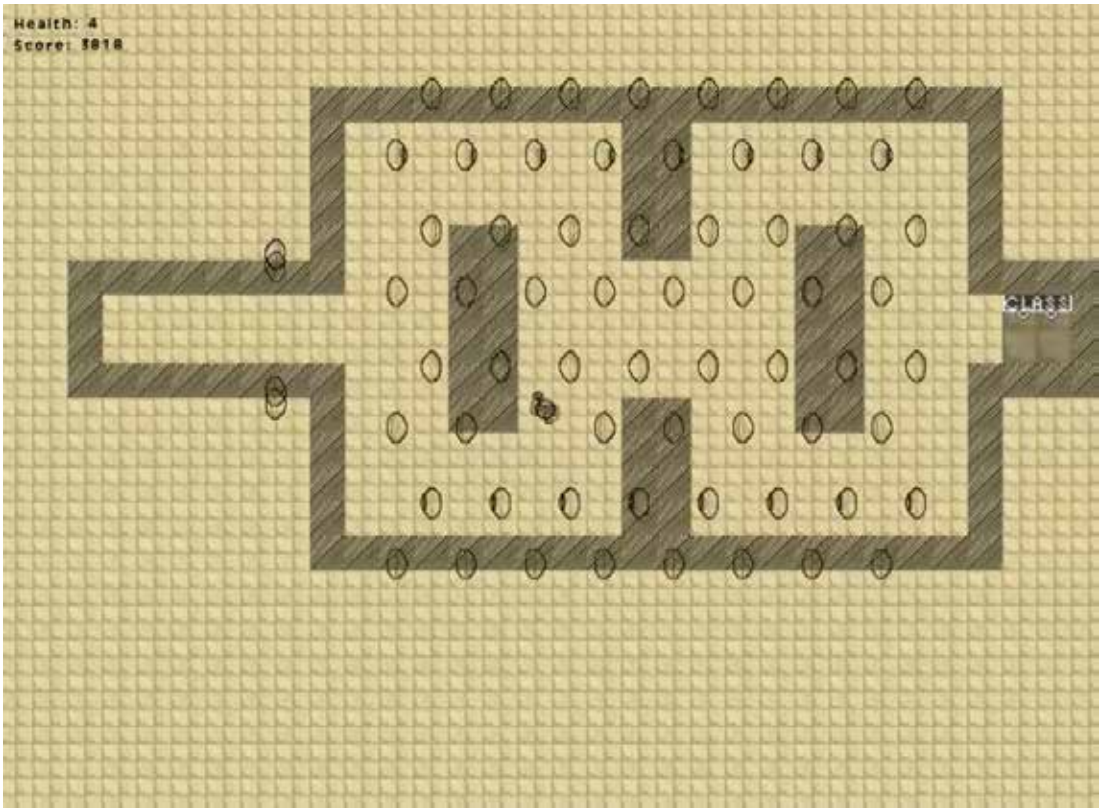

Figure 11. Academy mini-game.

5.4. Academy

ACADEMY, n. [from ACADEME] A modern school where football is taught.

Academy has the player navigating a series of hallways from a top down perspective, with American footballs flying across certain sections of the map (Figure 11). The player must get to class on time while evading the footballs, their health decreasing with each failed dodge, until they either reach the end or die. Their final score is based on the speed with which they manage to clear the game. For added cynicism, the class is a European football class.

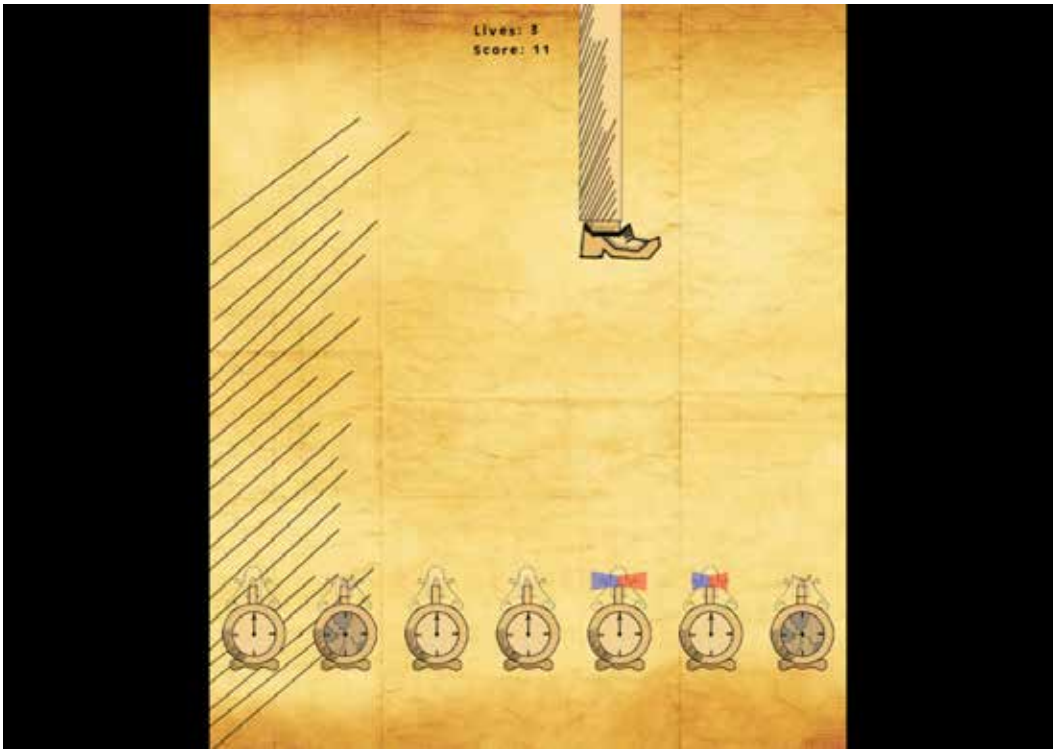

Figure 12. Coward mini-game.

5.5 Coward

COWARD, n. One who in a perilous emergency thinks with his legs.

For Coward, the player must stomp on a series of alarms as they attempt to go off (Figure 12). If an alarm is not stomped in time, it breaks, and if an inactive alarm is stomped, it also breaks, the player starting with a limited number of alarms that can break before losing. The pace of the game and number of alarms increases with time, to the point where it's nearly impossible to keep on top of it.

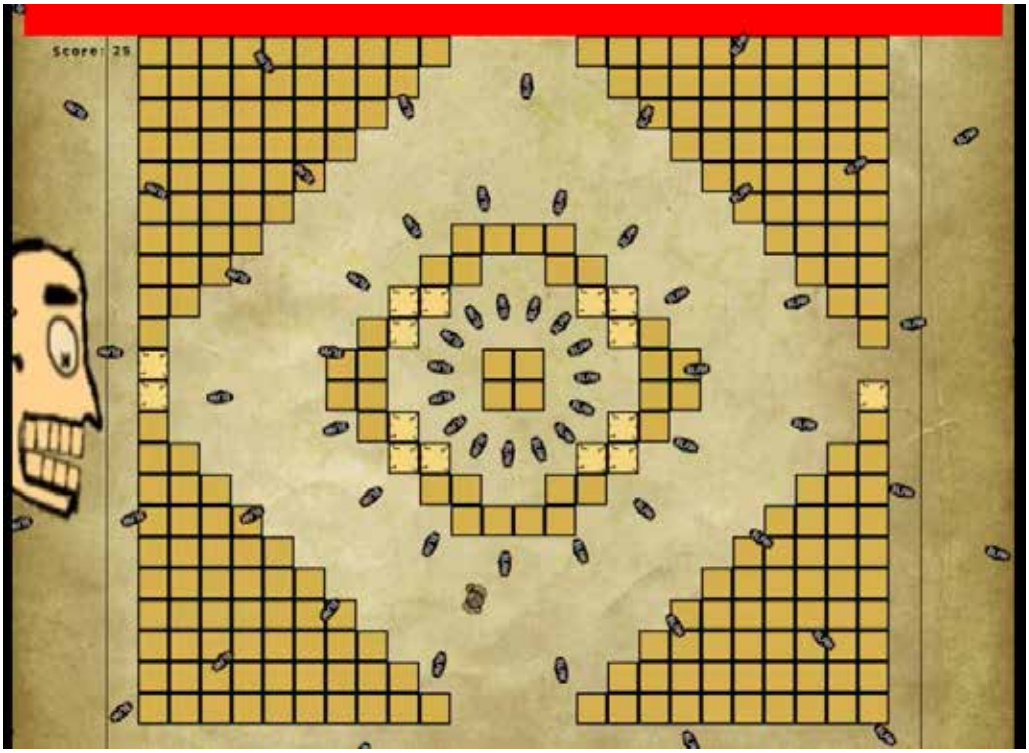

Figure 13. Bore mini-game.

5.6. Bore

BORE, n. A person who talks when you wish him to listen.

Bore plays like a retro shmup (gamer lingo for “shoot-‘em-up”) game, with the player evading the Bore’s projectiles and shooting through obstacles on the way (Figure 13). When the player approaches the Bore, the Bore switches the side of the screen is on, requiring the player to reach the other side again.

The projectiles used are variations on “blah” for the Bore’s blathering, and “um/uh” for the player trying to interrupt. Shooting the Bore directly lowers its health slightly, but the health will regenerate well before it would run out.

Figure 14. Art style samples.

6. Art style

Our artist originally set out on the process of asset creation planning on a pixelated, high-contrast art style, incorporating some aspects of ASCII art and an overall retro computer kind of feel to coordinate with the title of the game (at that time), *The Devil's Database*. That idea was dismissed before too long, as we concluded it didn't quite fit the tone of the game.

Based on some advice from our advisor, attentions turned to an art style based on older, earlier 20th century printmaking techniques, specifically linocuts, which could be physically made, printed, and scanned into the game (Figure 14). As interesting and attractive as this option was, it was determined to involve too much of a time investment, so we eventually settled on a style similar to that of newspaper illustrations from the early 20th century, typically characterized by hatched

shading, exaggerated figures, and (at least in the modern day) old, yellowed paper. The newspaper illustration style also happens to fit well with the origin of the Devil's Dictionary as a column in a San Francisco newspaper.

7. Game engine choice

The decision of what engine to use for our second MQP iteration was critical to our success We ultimately decided on GameMaker due to its ease of use, extensive documentation and overall flexibility. (Yoyo Games) We considered RPG Maker and Unity as other options, but these fell short in flexibility and ease of use, respectively. Had our final game selection been *The King in Yellow*, however, we would likely have gone with RPG Maker as its specializations would have been helpful for the more dialogue and atmosphere driven game.

Once GameMaker had been selected, the next bullet point to focus on was gaining experience with it, and quickly. Fortunately, the shift in project focus occurred shortly before winter break, giving an entire month of time to get to grips with the engine before anything concrete needed to be built with it. Our programmer had no prior experience with the Game Maker engine, and spent the break looking through the provided tutorials, as well as a few more complicated example games documented by the community. This proved helpful, as a few aspects of Game Maker are a little obtuse for how important they are, especially global variables.

8. Implementation issues

The first mini-game we implemented, Painting, was one of our programmer's first GameMaker constructions. As a result, the game was scoped to be mechanically simple, and yet implementing it proved to be a little more difficult than anticipated. The learning curve for GameMaker, though shallow, wasn't quite shallow enough to become proficient within the month between B and C terms.

However, the wealth of available resources and documentation on the engine proved to be an excellent mitigating factor for this potential issue.

The main technical issue encountered was the player controlled object not moving against walls, but instead stopping just short. This was fixed by carefully tuning the speed and player size so that the bug could never arise, while trying to work out a fix proceeding with other minigames.

Conversely, Litigation had no such issues, despite immediately following up Painting in the development process. Early testing showed that the starting positions needed to be randomized for the game to be challenging, and this was easily done by removing the objects from the Litigation game room and having the relevant controller object spawn them at a random height on the screen. When the final art was inserted, the acceptable heights needed to be changed slightly, as sometimes the objects would overlap the edges of the screen and get stuck there.

Longevity was another relatively painless process, as platforming controls for Game Maker are very well documented and nothing else the minigame tried to do was that complex. The biggest hurdle here was selected a random room to go to next on completion of the current one, but this was easily accomplished by running code to pick a random number and selecting a room depending on the result. Still, this was the first use of Game Maker Language (GML) thus far in the project, as everything up to this point had been done with the default commands. Our programmer would work more GML into future games, as it proved more efficient to use than the commands.

The menus may seem like a strange interlude to talk about here, but with a rudimentary grasp of GML, this was when most of the menu code was implemented. Up until this point, accessing levels was done by clicking on a specific range on the screen that was hard-coded in, with a placeholder object showing the location in question. Seeing as the design called for unlocking games by finding a word in a word search, there was a lot to be done for the menus.

The first problem run across was how to remove the hard-coded positions for the game selection buttons. Using GML to get the position of the object, as well as its height and width to see if the user clicked within that space only served to crash the game. Ultimately, an additional two invisible objects were added to the screen: one to follow the mouse cursor around, and one to be placed

wherever the user last clicked. These objects were then used to check for collisions with other objects, both solving the initial problem and making coding the word search unlocking much easier. The unlocks were handled by having pairs of objects, and unlocking the corresponding game if the user clicked on one while the 'lastclick' object was overlapping the other object of the same type.

The final major problem with implementing the menus was having the unlocks be persistent between plays of the game, let alone consistent between trips to the main menu. The solution ended up being fairly simple, however, as Game Maker's .ini files were both easier and more flexible to use than expected. Each minigame was given a section in the .ini file, with a field for if it was unlocked, as well as the high scores and player names associated. Updating the score list after each game completion was easily done as well, since the scores were already known to be in a sorted state. That meant that inserting a new score was just a matter of bubbling through the list until finding a higher score, and stopping the iteration there.

Compared to the menus, Academy was not a problem. Dynamically setting the football launchers initial direction had a couple bugs, but these were sorted out by 4 generic objects that would set the direction of certain objects overlapping them on room creation. The initial approach had the launched footballs themselves have their directions adjusted by these control objects, but that ran into a problem where already launched footballs would have their directions changed, creating impassable walls of footballs.

Coward was an interesting exercise, as nearly all of its implementation was within GML! There were a few days spent just thinking about how to trigger the alarms in the game, and to make sure that only visible ones could be selected. Eventually, pseudocode was worked out and implemented into the game, and it mostly worked perfectly! The key insight was to have each alarm have four different possible sprites, each representing a possible state. These states were inactive (0), broken (1), active (2) and off-screen (3). This worked, until the animation was added. The indexes for 'active' were stretched from just index 2 to 2-13, and this created several new bugs. Alarms that were off-screen immediately activated on being placed onscreen, the alarms would never go off on their own and break, and the game would even crash if there were no inactive alarms. These all ended up being

simple fixes once the bug was tracked down, thankfully, by changing an index, or adding an additional check to make sure an inactive alarm existed. Most interestingly was that setting the timer to 60 frames wouldn't have the alarm break, but setting it to 61 frames would. It is suspected that this was a race condition, and the index check occurred with index 14, before index 14 set the index back to 2.

For the final game implemented, Bore, the process actually went smoothly for the most part. Our programmer worked out which object needed which code, planned out the required control objects, and mechanically everything worked out perfectly. However, when it came to setting up a new wall pattern, a serious issue was run into with functionality that Game Maker simply lacked: a quick way to manually set the contents of an array. For a 2-dimensional array like the one used here, rather than "=[(value, value), (value, value)]" each individual cell needed to be set with "array[x][y] = value". For the 24x22 array used here, even with initializing the array to the most used (null) value of zero, it would still take over 100 individual lines of setting. Instead of storing the map as an array, the next approach tried was 1 dimensional arrays of X and Y coordinates of objects. This would have worked with less lines than the previous approach, but at around 70 was still far too large to scale to multiple maps, and was even more error prone.

The next attempted approach was to store the maps as text files, as Game Maker could read those in at a low impact. This worked flawlessly, and allowed for quick and easy map prototyping and edits: even easier than using Game Maker's room editor. Had our programmer known this sooner, this method of map loading would have been used for Academy and Longevity as well.

9. Testing

Due to the rushed development near the end of D Term, we did not have nearly as much time for testing as we would have liked. However, what little testing was done was quite beneficial, and nearly all feedback was positive. A few of the maps for Longevity and Academy were tweaked, the word search was subtly altered to remove a duplicate word, and the difficulty of Painting was lowered slightly. Bore, Litigation and Coward were unchanged as a result of this testing.

Testing was done through Skype with screen-sharing, and the first tester played for over an hour. Default high scores were shown to be difficult to match without some effort being put in, and the overall difficulty and theming of *The Devil's Dictionary* showed through. The game also showed itself to be self-explanatory, as we did not have to assist the tester in understanding what needed to be done.

Overall, the testing process, though brief, confirmed that *The Devil's Dictionary* was in a mostly finalized state, as all difficulty and polish related feedback was for aspects of the game that were already noted to need minor tweaks. The game was thoroughly enjoyed, and so may be considered a success.

10. Conclusions

10.1. What went right

While relatively little of *PermaDeath in Venice* actually reached a fully functional state, we made some excellent progress regarding the game design. In particular, we think our level design methodology of sectioning the maps into the two paths, the linear, time-based main paths, and the free-form, spatial side paths, was a good idea, and a nice way of conveying the distinction between Aschenbach's structured, orderly life, and feelings of passion and disorder introduced by his infatuation with Tadzio that formed the backbone of the novella.

In addition, while the some of the design details for permadeath weren't completely worked out, it would have been a very interesting and novel interpretation of the permadeath concept compared to previous uses, turning the target away from the player, and towards the game itself. Also, though there were certainly some rough patches in the beginning, once the capability of directly implementing in-game levels, complete with pathfinding, multiple layers, etc., as image files was in place, putting levels into the game became much simpler than it might have been otherwise, as well as streamlining the process of iterating maps, as an adjustment to the level geometry would just require an adjustment to the image file, as opposed to making said adjustments by going into and editing the code itself.

10.2. What went wrong

Of course, for everything that went right with this project, other problems seemed to crop up. *PDV* in particular at times felt less like a project and more like a series of obstacles to overcome.

If we had to do this project all over again, more time would have been spent prior to beginning the project familiarizing with the technology of the game engine being used, as a decent amount of time was spent getting to grips with two engines over the course of the project. The initial scope of the project would also be more constrained, as *PDV* seemed to grow exponentially in complexity. However, if we had a third member on our team – preferably someone already familiar with Perlenspiel – the scope would have been reasonable, albeit challenging. Source control would be established from the very beginning, but instead of using GitHub or BitBucket, we would use Dropbox for instant file sharing and a centralized location.

For *The Devil's Dictionary*, more time would have been spent designing specific games and in greater detail than we originally designed them, as there ended up being miscommunication on some aspects of games that had to be discussed later, delaying us a little farther. The final scope was a little larger than expected, but thankfully the design we chose specifically allowed for cutting pieces of our game without losing the whole. Finally, the additional knowledge on design and implementation time would have us start the polishing process sooner, so that we could get useful testing feedback with more than a couple weeks to act on it.

Works Cited

Bierce, Ambrose. *The Devil's Dictionary*. New York: Dover Publications, 1958. Print.

Yoyo Games. "Game Maker Language Documentation." GML Overview. Web. <https://docs.yoyogames.com/source/dadiospice/002_reference/001_gml language overview/index.html>.

Mann, Thomas. *Death In Venice and Other Stories*. Trans. David Luke. New York: Bantam Dell, 2008. Print.

Moriarty, Brian. "Perlenspiel | Home." *Perlenspiel*. Worcester Polytechnic Institute. Web. <http://users.wpi.edu/~bmoriarty/ps/index.html>.

Appendix 1: Asset List

ART ASSETS
MAIN MENU
start screen
place to enter name
Definition icons-clickable
Definition icons-locked
word search
PAINTING
Canvas icon
Paint Icon
Weather Icon
Speech Bubble (multiple)
Critics
Background
LONGEVITY
Player Character
Death wall
Pickups
Terrain Tile
Background
LITIGATION
Cannon Icon
Pig Icon
Court Icon
Sausage Icon
Background
ACADEMY
Player Character

Football Icon
Wall Tile
Floor Tile
Goal Icon
COWARD
Emergency Box Icon (not going off)
Emergency Box Icon (going off)
Emergency Box (broken)
Legs (Player Character)
Background
BORE
Player character
Enemy head
Various words
Player shots
Destructible wall
Non-Destructible Wall

Asset sources

All in-game sounds, unless otherwise stated, were obtained from freesound.org.

All reference art, unless otherwise stated, was obtained through Wikimedia Commons.

Appendix 3: Design Notes

Permadeath In Venice design notes are incorporated into Section 2, Design Issues.

The Devil's Dictionary

The following consists of a combination of Design and Implementation notes, pulled directly from the programmers Pastebin.

Bore

Bore design with fleshed out object list
BORE: head shoots words
you dodge words and get to head
get close, head switches side (yay, score!)
small hitbox? large hitbox? I don't know!
Grinning Colossus style enemy HP
Wall pattern has multiple variations it switches between
get hit at all and it's over

Object List
Player
Arrow Keys : move (can't though either wall)
Space : fire
Player Bullet
moves in direction it was fired
mutual destruction with destructible wall
hits boss, hurts boss
hits wall, disappears (walls off-screen)
Boss Trigger

Player collides, it switches sides and tells the boss to
Adds to score
Deletes all walls/destructible walls and spawns a new set

Boss
Regens health proportional to how much is missing
Tells bullets to spawn
Boss Bullet
Kills player on contact (next room)
Indestructible Wall, Destructible Wall
No code required

Coward

Coward: RPG system where the only option is to run, active battles allow evading all enemy attacks, survive as long as possible
Coward game- maybe the coward is the enemy? That results in a 'win' though.
COWARD GAME DESIGN
OLD DESIGN
possibly RPG style where the only option is to run (but it fails several times before working- or starts at 20% chance and rises by 20% every fail)
must survive the enemy (active battle in some way) for a few rounds before the game shifts to a top-down overworld type thing
must run away from the enemy (amass score for surviving here)
Eventually get caught and the cycle repeats (enemy is faster than you)

FINAL DESIGN
left/right to move
space to stomp on alarms
stomp alarm going off = +++points, it stops
stomp alarm not going off = it breaks, -1 life
starts with 5 alarms / 5 lives but more get revealed as it goes along (up to 11/13)
rate of alarm-ing increases with the time as well (20-40% of all active alarms?)
failing to stomp an alarm going off within a few seconds = it breaks, too

Litigation

Litigation design: cannon that fires pigs into a courthouse which makes sausage.
Both are moving vertically, player must time the shot so that it hits the courthouse.
After X misses, game is over, score is number of successful hits.
With each successful hit, the speed of the objects increases.
Player may or may not be able to move, depending on how it plays.

Longevity

Longevity: some sort of endless runner/tower climb (?) with pickups to extend the time
Purely timer based, mistakes subtract from the timer.
Think Canabalt + Yoshi's Island autoscroller with Sonic level design. (Canabalt scrolling, can be out run, pickups out of the way, better rewards higher up)
Longevity design:
Variously designed rooms (wider than the screen)
screen scrolls, on reaching end area load another
have a deadly obstacle behind the player that moves at ~2/3s of the players speed
standard platform controls (move, jump)
Player dies on touching singular deadly obstacle or running out of time.
Gains score from score pickups and surviving, gains time from hourglasses
controls
gravity
left/right
jumping
scrolling follows player
timer goes up on collecting hourglass (send back death wall)
goes down with time
on 0, game ends
deadly obstacle (acts as mechanics-based timer)
deadly wall pursues player at 2/3s their max speed (?)
on hit, game ends
remember distance behind player between rooms

track room count in lives, increase speed with this to a point
have a maximum lead (always onscreen?)
score
goes up 1 point every 0.2 seconds
goes up 25 for a score pickup (out of the way)
rooms
have a pool of X rooms
on leaving right side of screen load a random different room from the pool
remember the player Y position and speeds and preserve them

Painting

Game: Move a canvas around to catch falling paint, avoiding the weather and dealing with obnoxious critics. Movement can be done via keyboard or moving the mouse. The higher the paint is caught, the higher the score for catching it. The intensity ramps up with time until eventually, failure is inevitable. Scene of critics viewing the 'masterpiece' afterwards, which contains the final score. Different weather patterns can include rain (acts like bad paint), wind (comes from the sides) and lightning (forecasted and then fills its space instantly). Critics may walk up to the player and cause distractions, block movement or cover part of the play area with dialogue.
Critic Implementation: have a speech bubble cover a large part of the top of the screen.
Difficulty scaling: The difficulty should ramp up gradually over time, after about a minute should drop at a rate of one every frame?

Appendix 4: IRB Protocol

Purpose of Study: To obtain feedback on our game in order to make improvements and adjustments as suggested by the collected data.

Study Protocol: Participants come in, and are provided a computer or other device on which to play the current build of the game. The present investigators would observe said participant for their reactions during game play, while metrics built into the game would collect data about participant's playtime, as well as the playthrough itself being recorded via screen

capture. Afterwards, participant would be asked to fill out a short questionnaire to ascertain their subjective experience playing the game.

Opening Briefing for playtesters: “Hello, and thank you for volunteering to test our game *Permadeath in Venice*. Before we begin, could you please sign this form? [tester signs Informed Consent Form] Thank you. During your playtest, the game will be recording a variety of metrics about you play style, as well as screen capturing your playthrough. Once your playthrough has concluded, we’ll have you complete a brief post-game survey. At no point during this test, or the survey after, will any sort of personal and/or identifying information about you be recorded. Please begin playing when you feel ready.”

Metrics to be recorded-

- Total playtime/playtime of various segments of the game
- Total amount of clicks/how many clicks does the player use to move around
- How many relevant clicks does the player make (clicking somewhere that facilitates movement as opposed to clicking somewhere that doesn't)
- Amount of idle time (how long does the player spend not moving/not interacting with the game)

Questions for post-test survey-

- Which part of the game would you consider the strongest/most effective? Which was weakest/least effective?
- Were there any other parts of the game that stood out?
- In your opinion, what was this game about/what was the point of the game?
- What, if any, sort of emotions did playing this game evoke?

Did you feel like the atmosphere (art/music/etc.) of the game contributed or detracted from the overall experience?

Did the game seem to have a narrative of some sort? What do you think said narrative is?

Did you feel like the game was too long, too short, or an adequate length?

Did any portions of the game frustrate you? If so, which one's and why?

What are the rules and objectives of the game? How did you discover them?

How difficult did the game seem?

Did it consistently hold your interest?

What would it have been good to know before you started?

How would you describe it to someone who has never played?

At what point did you feel truly invested in pursuing the gold bead? Did you ever feel invested in the pursuit?

Do you have any comments or questions that you believe weren't addressed in one of the preceding questions?

Appendix 5: Playtest Notes

General feedback:

Music was an excellent choice.

The definitions are a nice touch.

The ability to place an excessively long name is appreciated.

Word search:

Was quickly understood, but slowed the pace.

Only took around 3 minutes to unlock everything, though.

Unlocking wasn't clear (a sound effect would fix this)

A sound effect for this was already on the todo list and was noted that the build lacked many.

Games:

Academy: There's one map that is MUCH more difficult than the rest.

Very fun otherwise.

Bore: Another fun game.

Coward: Another very enjoyable game.

Litigation: Heavily enjoyed.

Longevity: Map 3 needs to be nerfed, the bonus on map 4 needs to be nerfed. The leftward movement that is forced in a few places is great, along with death's approach.

Painting: Needs a slight difficulty nerf, but was a lot of fun.

What would you change?

Sound effects on the word search.

Observations:

One Academy map needs an additional blocker.

Bore was in the word search twice.

Bore aimed bullets: first spawn is on the wrong side.

Bore needs something to block the very bottom. (another health bar)

The Litigation judge needs precise masking.

Longevity needs to prevent the same map from showing up consecutively.

Longevity was most played, followed by Bore, Academy, Coward, Painting, Litigation.