Chaotic Mind - Final Report

“Spinodal Studios”

Chris Cooper, Carey Metcalfe, and Alex Szczuczko

Table of Contents:

Executive Summary

Chaotic Mind is a top-down shooter set in a procedurally generated maze, with the goal of collecting a number of objects in each level. The player is granted the ability to manipulate the maze by sliding around large portions of it in the form of tiles on a grid.

The player also must defend themselves from enemies through the use of weapons and offensive/defensive employment of their tile shifting ability. The game is two dimensional and has a broad target audience due to the juxtaposition of contrasting gameplay elements. Low to high end Windows PCs should be able to play Chaotic Mind.

Our primary selection of music has been approved by the author for use in the game, provided it remains non-monetized.

The streamlined User Experience for a one level game means the UI requirements are simple, consisting of the three start and end screens, main game screen, as well as the tile shifting overlay. Controls consist of keyboard and mouse, mapped to the standard WASD movement and mouse aiming functions.

Characters are differentiated through the use of top-down perspective sprites, their movement and their weapons.

The story is not readily apparent in a single level game, however in a hypothetical multi-level game it would consist of a single tension arc that is modulated by the difficulty of the game. The Game World, though largely transparent to the player, is an abstract representation of the protagonist's mind.

All art assets have been produced in house, with music and other audio assets being sourced externally from approved or royalty-free sources. Animated sprites exist in combination with static and dynamically rendered sprites.

Playtesting was conducted in two large segments on both a paper prototype and an early digital prototype.

Part A - Game Design

1 Vision Statement

1.1 Logline

Chaotic Mind is a top-down shooter set in a shape-shifting maze.

1.2 Gameplay Synopsis

A multi-level game is set in a series of progressively harder grid-based maze-maps that the player must navigate through by shifting squares of the map around and defeating any enemies that they encounter. The player must collect a number of relics from the protagonist's past to progress to the next level.

At any given time, the player will be engaged in a set of activities. There are two separate gameplay considerations that the player needs to be thinking about while playing the game. The

first is the high level game. The player will be trying to shift the maze and open up a path to the next goal object. The second gameplay mechanic will involve things that would typically be found in a top-down shooter such as dispatching enemies, managing health and other resources, and planning where to go next.

Not at all times, but quite often the player may need to defend themselves from enemy attacks, which could be accomplished through the use of weapons or carefully timed tile shifts. The player will be granted the ability to selectively pause the game in order to have time to consider the potentially complex set of information.

2 Audience and Platform

2.1 Target audience

The appeal of this type of game is quite broad. Top-down games are generally accessible due to the 2D environment, which requires the user to perform less advanced space processing than in a 3D environment.

Shooters do require a degree of reflexive skill (rapid response to targets, avoiding projectiles, etc.), and Puzzle games require careful thought. The contrast available in Chaotic Mind between real-time shooting and atomic-time shifting allows a flexibility in which skill-set a player would like to use more.

Knowledge of English (or any language) is not required to learn the game, allowing a global geographic scope to be targeted. No specific age or gender is targeted.

2.2 Target platform

Windows PC is the production platform. Consoles are not targeted. Although top-down “dual-stick” shooters are well established on consoles, the goals of this course meant the extra development and testing required for console and controller support were not worthwhile.

2.3 System requirements

Any version of Windows since (and including) Windows XP can run Chaotic Mind. The game is not particularly CPU or GPU intensive, however XNA requires a graphics card that supports Shader Model 1.1 and DirectX 9.0c.

3 Legal

We have obtained authorization to use music from Rain (Phutureprimitive) in Chaotic Mind, provided that the game remains non-monetized. Any future monetization will require discussion of compensation with Rain, if his music is to be included in the monetized game.

4 Gameplay

4.1 Overview

The uniqueness of this game stems from that fact that it’s a mash-up of different genres of games. The top-down shooter aspect is augmented with the secondary gameplay mechanic of managing

the shifting maze. Introducing this mechanic allows the player to use many more styles of play. For example, instead of the typical approach of trying to gun down any enemy in your way, this game allows for other play styles such as the more stealthy, passive tactic of luring enemies into a room, then shifting the board such that they get trapped and can't harm you.

4.2 Gameplay description

Gameplay will follow the same flow that is documented elsewhere in both this document and in the project proposal. The player will navigate the maze trying to recover objects (relics of his past) and avoiding (or destroying) enemies that get his way. The initial prototype of the game is shown below. In this image, the "fog-of-war" effect is demonstrated, obscuring the player's vision.

4.3 Controls

Interfaces: The player will use 3 interfaces while playing the game. First is the main menu of the game. This will have a soft background track playing and will have a prompt to click the mouse to begin. From there, the player will be placed in a random level of medium size. The in-game HUD is fairly sparse, as to not distract or confuse the player.

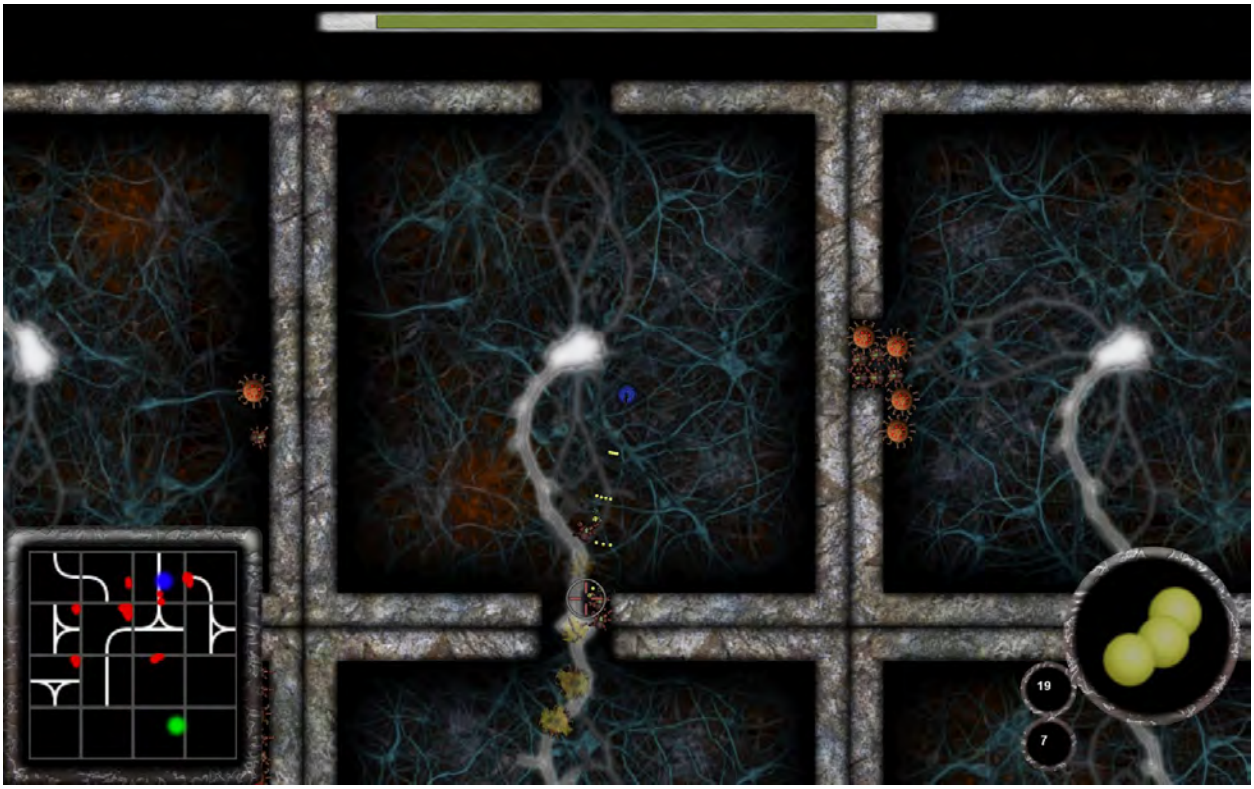

Figure 4.1.a - Screenshot of the player shooting at some enemies

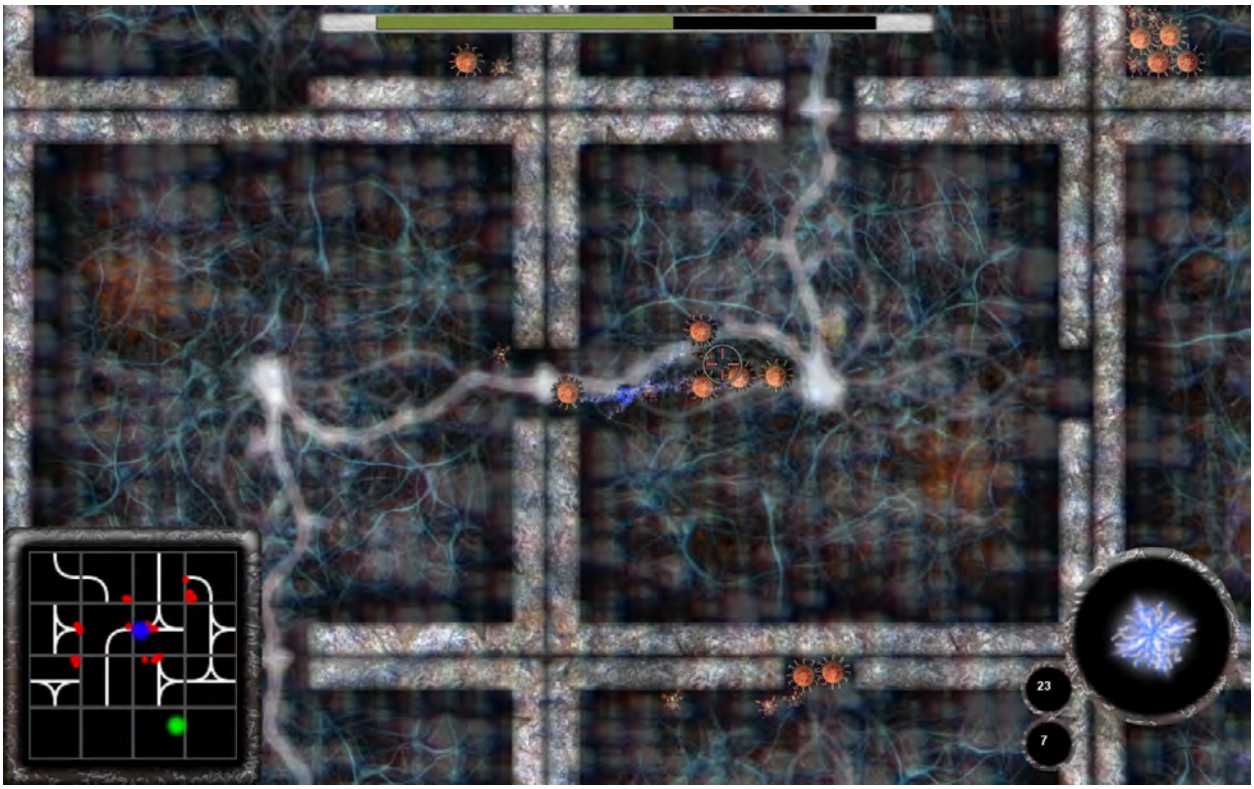

Figure 4.1.b - Screenshot of visual feedback for the player being attacked and damaged by enemies

There will be a “sanity” gauge that shows the player’s sanity (analogous to health), a minimap that shows a representation of all the explored tiles, and a “shift” button (mapped to spacebar) that, when pressed, shows the map shifting overlay.

When the player is viewing the shifting overlay, the game is paused. From the overlay, the player will be able to see the entire map (the unexplored sections will still be blank) and buttons around the perimeter the map, one on each side per column and row. Clicking these buttons results in a random new tile being pushed into the corresponding position, shifting the entire line of tiles. The tile that falls off the other side fades out and is destroyed.

Rules:
Player - The player must avoid getting hit by the enemies, as this will decrease his sanity meter. When the sanity meter is depleted, the player loses the level and is sent to a death screen, and finally returns to the title screen. The player is able to move, shift tiles, and attack enemies.

Tiles - There are also a few rules that govern the tile shifting mechanic. While it is possible to shift the row of tiles with the object you are currently trying to reach on it, the game will discourage the player from shifting the goal tile off the map. When the player does so, they will lose a portion of their sanity, and a new random location is chosen for the goal.

Enemies - Enemies will not be able to initiate any tile shifts. They will continually be trying to swarm the player. They will have to walk around the maze to get to the player.

Scoring/winning conditions: There won’t be any primary scoring condition, this is a game where the only real objective is to survive and complete the level. The winning conditions of the game align with the plot of the game in that it is necessary to collect all the artifacts in the level and defeat the boss to progress to the next level.

4.4 Levels

Levels are procedurally generated grid-based mazes. In a square grid of variable size there will exist one square tile per grid cell. Level difficulty is mostly a function of the size of the level, as more shifts and thought are required for completion. A beginning level would be in the size range of 2-4 side-length, and more advanced levels would reasonably have approximately 5-8 side-length, but could conceivably be very large (beyond 10 side-length).

Each side of a tile has a doorway located in the middle, through which characters and projectiles may pass if the doorway is not blocked. Doorways are blocked as part of level generation, creating permutations of tiles with between 0 and 3 doorways blocked. See figure 4.4.b. The overall status of a doorway is dependent on the relevant neighbouring tile. Both doors must be open for a doorway to be passable. See figure 4.4.c.

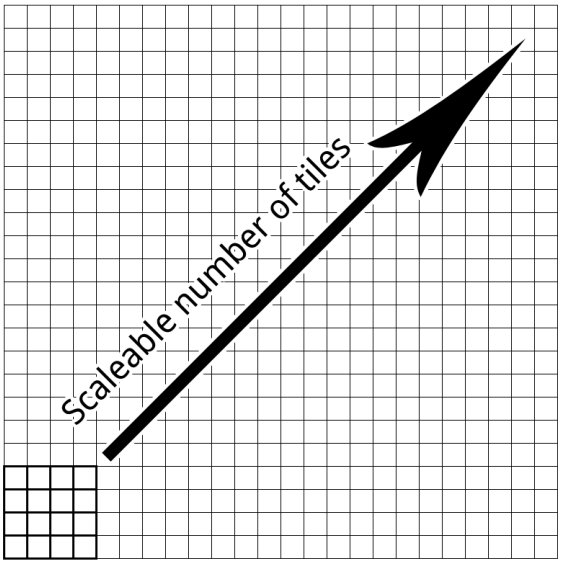

Figure 4.4.a - Level grid size range

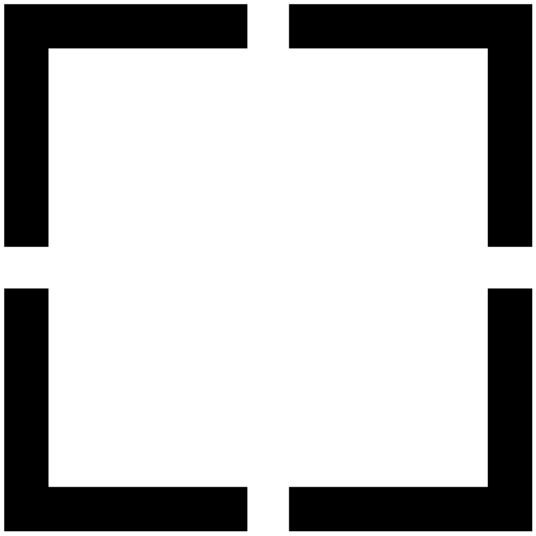

Figure 4.4.b - Single tile, 4 open door layout

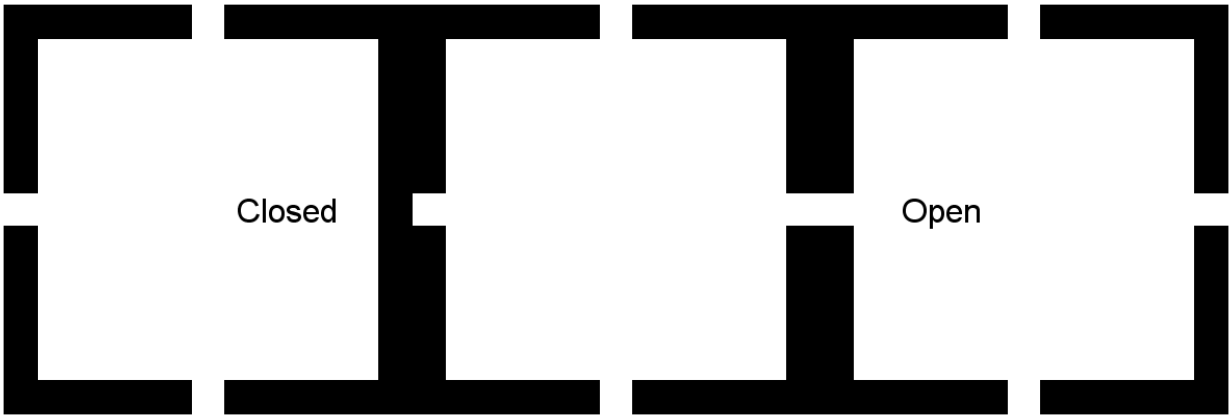

Figure 4.4.c - Door overall open status is neighbour-tile-dependent

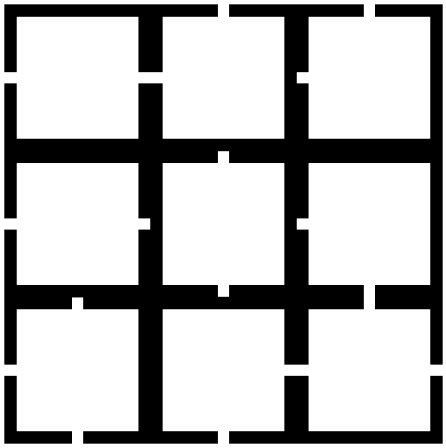

Figure 4.4.d - Example doorway open/closed pattern

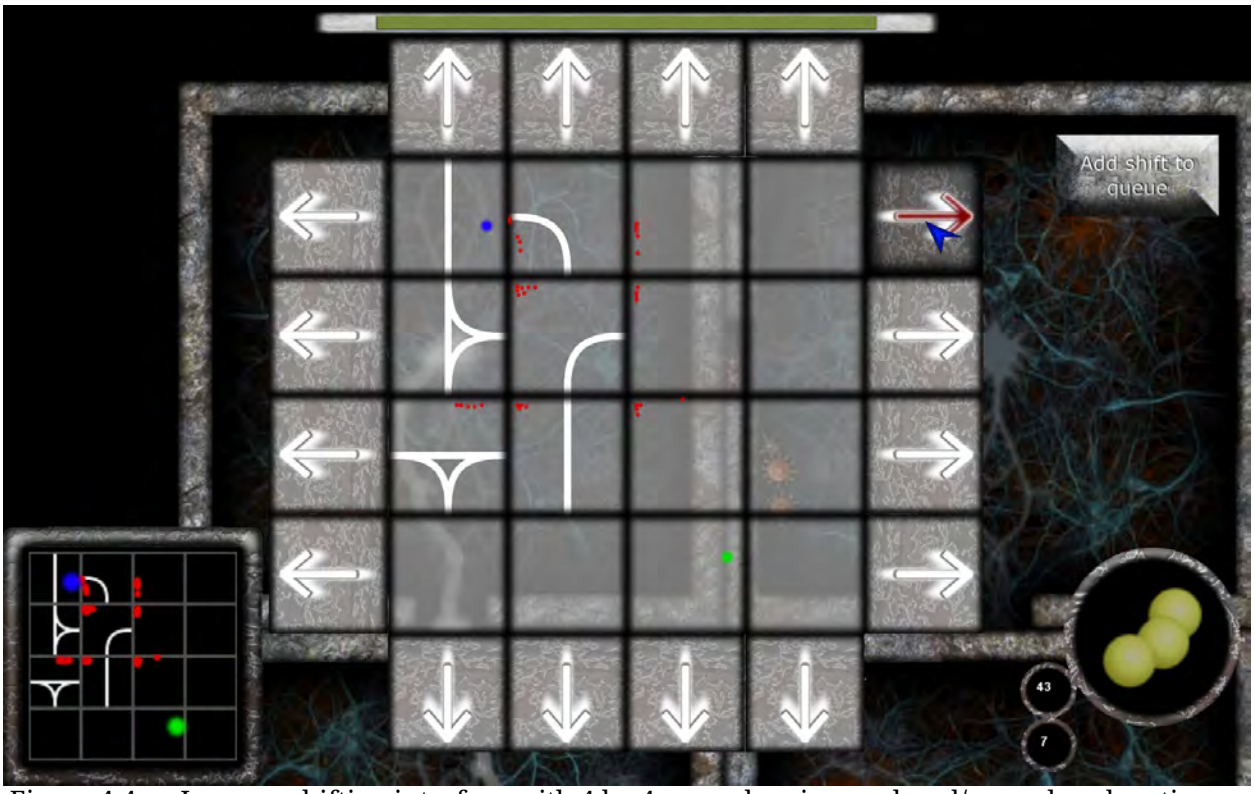

Figure 4.4.e - In game shifting interface with 4 by 4 map, showing explored/unexplored sections

4.5 UI/UX flowchart

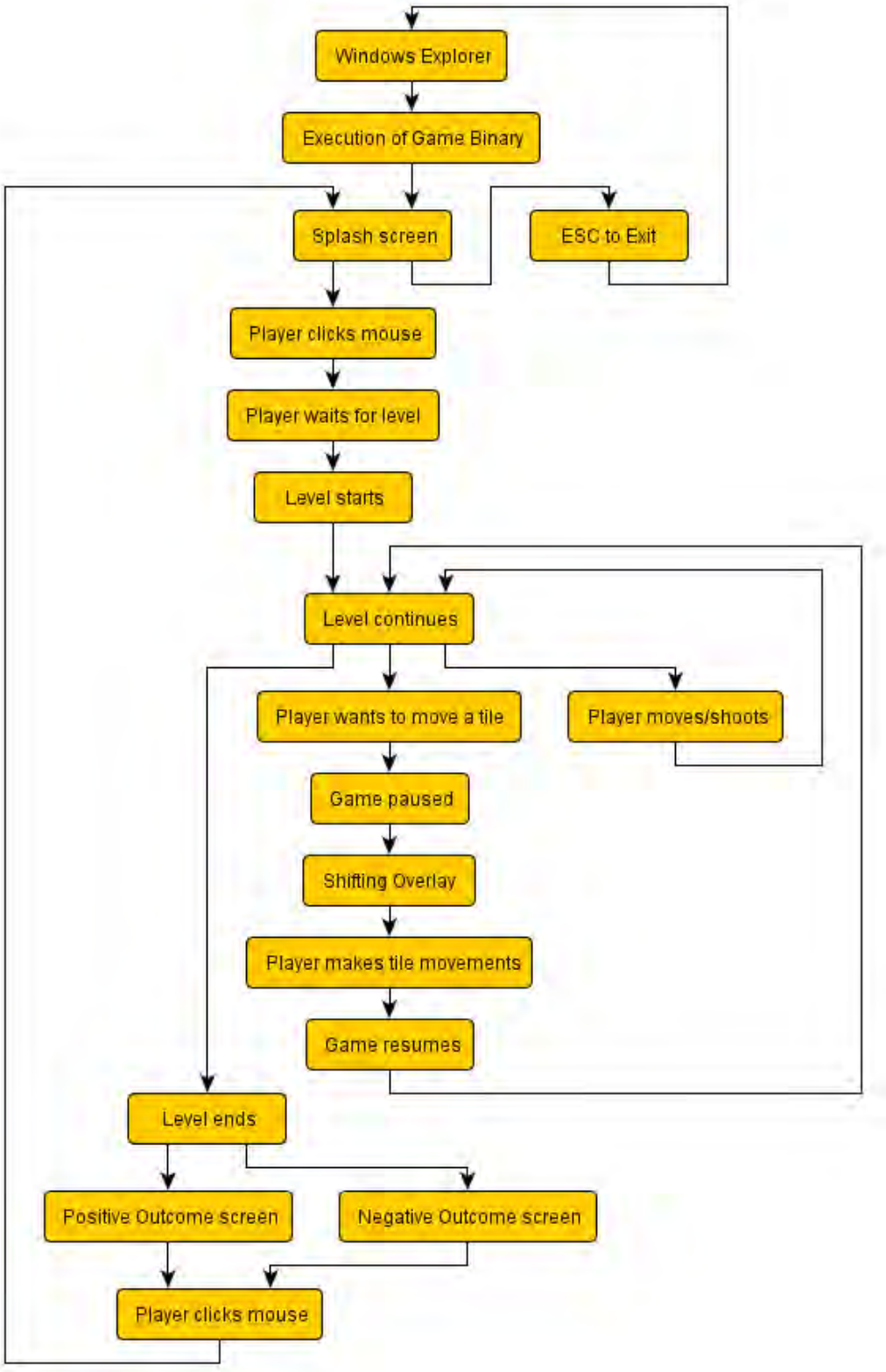

Figure 4.5.a - Flowchart of user experience playing the single-level game

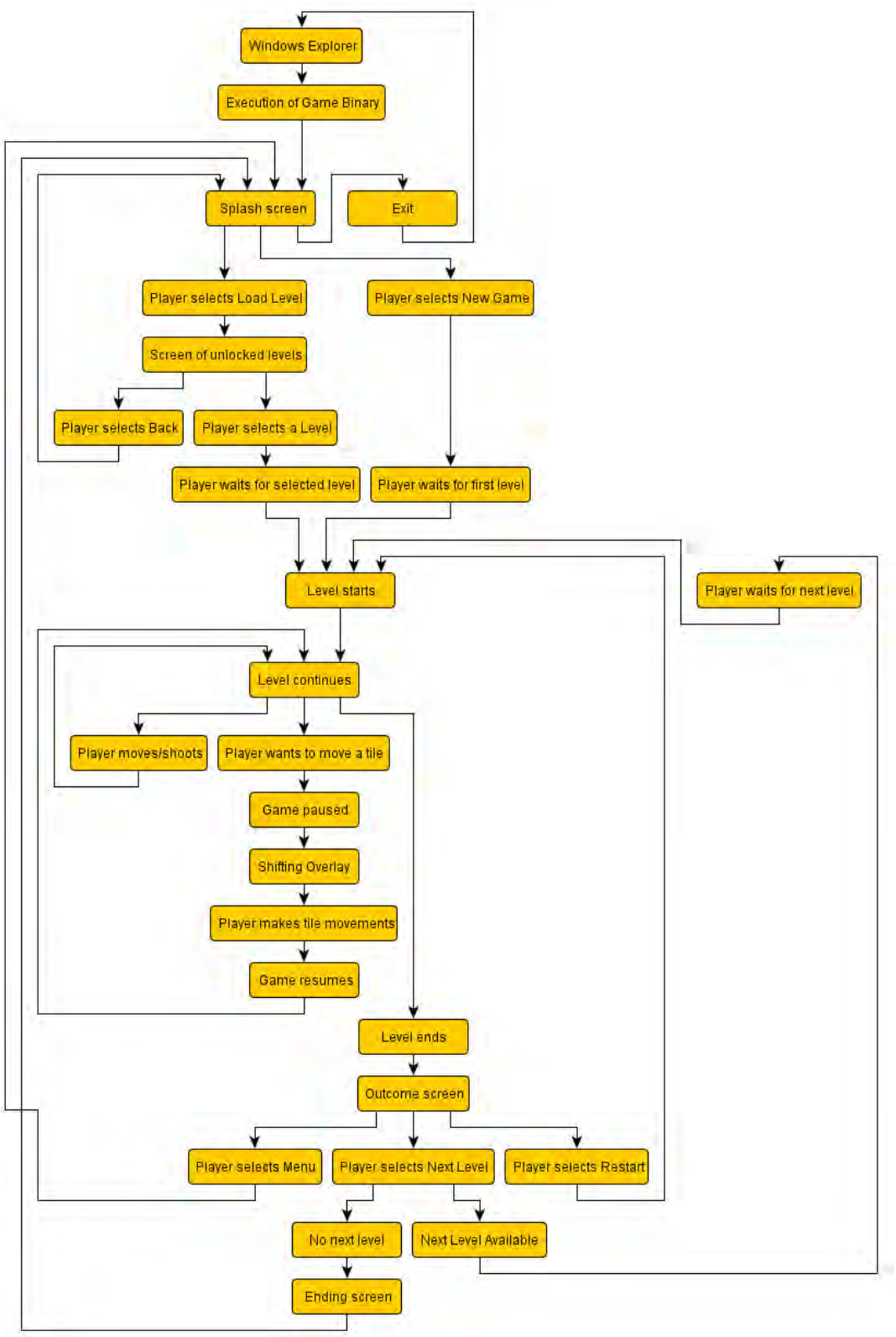

Figure 4.5.b - Flowchart of user experience playing a hypothetical multi-level game

5 Game Characters

5.1 Character design

In-game description and characterization is performed entirely through sprites, their animations, and their movement behaviours. All non-player characters are hostile to that of the player.

5.2 Player Character

The player character has a non-humanoid appearance. From a top-down perspective the body of the character is a patterned circle. The direction that weapons are pointing is shown as a rod shape, starting from the centre of the circle and moving through one edge. The player character's movement is highly responsive to the player's control inputs. See figure 5.2.a.

5.3 Non-Player Characters

The two enemy types are constructed with an arthropodoid appearance. They track the player and attack when they come into range. Their movement is randomized to varying degrees, which adds to the persona of the particular enemy. On death they have an animated splatter sequence before fading out. See figure 5.3.a.

Figure 5.3.a - Enemy death sequences

Swarmers are small, quick enemies that look something like a tailless scorpion. Their method of damaging the player is a melee lunge. After an attack, they back off slightly from the player before lunging again. See figure 5.3.b.

Parasites are larger, circular enemies with centipede inspired legs around their perimeter. They move slower than swarmers, but are more dangerous to the player if they are ignored, since they can easily corner the player. The parasites' attack is a bolt of lightning, which is limited in range to about three of their diameters. See figure 5.3.c.

Figure 5.2.a - Player

Figure 5.3.b - Swarmer

Figure 5.3.c - Parasite

6 Story

6.1 Overview

The game's story is fairly simple. The player enters in as the protagonist starts to combat the mind virus antagonist. If the game were expanded beyond one level, the initial levels would form a tutorial for the player in step with the protagonist's learning about his mind.

Tension picks could increase as the the levels become more difficult and the player encounters new and more varied enemies.

The climax comes in the final level of a multi-level game, where the player must apply skills they've learnt previously to defeat the mind virus final boss.

6.2 Backstory

"You are a brilliant physicist plagued by a terrible virus of the mind. The madness haunts your dreams and is siphoning off your thoughts and memories. You've already forgotten the first few decades of your life and will forget everything you have ever known if you don't take action. The only cure is to delve deep within the maze of your own mind and rediscover your past, ridding yourself of the madness which has wormed its way into the deepest and darkest corners of your mind. You enter a deep meditation and and confront the nightmares, wondering if you'll ever wake up again…"

7 Game World

The game world is simple, given the scope of the project. Basically each level is an abstract representation of a part of the protagonist's fragmented brain.

These world foundation concepts of thought and jumbled pieces is communicated through two primary methods. The first method is the gameplay mechanic involving sliding disjointed pieces resembling clumps of neurons, and the second is the music, which was selected for its iterative beat/melody.

The protagonist's mind -- and thus the world -- is under stress, and this is communicated through the shooting aspects of gameplay, and again, the music.

8 Media List

Interface assets:

- x3 Menu screens (Start, Death, Win)
- x4 Tile shifting overlay icons (U,D,L,R)
- x3 Doorway state indicators (vector)

Environments:

- x1 Tile background
- x3 Doorway state indicators (nerve-like)
- x1 Collectable item

Characters:

- x1 Player Character

- x2 Enemies

Animation:
- x2 Projectile variants
- x2 Enemies
- x2 Enemy deaths

Audio:
- x10 Songs from album *Kinetik* by Phutureprimative. Each approx. 5 minutes in length.
- x1 Song *Predatory Instincts (Elevator Mix)* by Phutureprimative. 2:24 in length. (For title screen)

Part B - Production

1 Implementation

The final implementation included almost everything that we had originally planned for. We were able to build a very solid and quite easy-to-work-with framework, and by late in the development stage, adding new content was extremely easy. With just a little more time (and perhaps a real artist), we could have added many more enemies, weapons, and collectable types. We avoided this for now, keeping in mind our goal of producing a polished and fun game, not necessarily a content-rich game.

The most complex area to implement was also the most interesting; that is the shifting mechanic and interface. The most difficult part was ensuring that object in the rooms moved with the tiles, instead of being pushed along by the walls. Originally, tiles were reponsible for sifting through an entire list of objects, finding the ones that were inside them, and applying a shift to them. This implementation worked, but was hard to modify (all objects had to be in a single array) and both memory and processor inefficient (creating multiple arrays, going through them multiple times). However, after a few technical hurdles, we developed a more elegant implementation, wherein each object in the game queries the map manager for the presence of a shifting tile, and after determining if it is inside one, applies the correct amount of velocity to itself by again querying for the current speed of the tile. This has the effect of keeping objects stationary relative to the tiles, even when they shift.

A feature that made it into the codebase, but not into the gameplay that we showed off at the demonstration was using raycasting to implement projectiles that traveled instantly (which is more like a real weapon). We implemented this by reusing as much code as possible from the normal method of shooting. Instead of giving the projectile a velocity and letting it move to the target, we used a raycasting function to spawn the projectile immediately at the place it would have hit the enemy, and let the original collision detection of the projectile handle the damaging of it. The reason this didn't make it into the demo is that, during playtesting, we found it difficult to tell when you were actually hitting the enemies, because there was no visual confirmation, except when the enemy died. This visual indication may be added at a later date, so we can use this feature.

One feature that we didn't implement due to time restrictions was a final boss of each level. We also did not implement different levels, since we designed the game around the concept of 5 mins of gameplay, and there was not enough time later in the development to integrate levels.

Music implementation was complicated by the poor support for long audio tracks and self-made playlists in XNA. The XNA MediaPlayer class provides a very rushed interface and codec support

through a hacky extension of Windows Media Player libraries. Custom implementation of a playlist, loader and control functions was necessary for success.

Sound effects management started as a stripped down version of the music loader class. In the end, sound effects were divided into two sections that used two different classes: SoundEffects and SoundEffectInstances. SoundEffectInstances can be played, stopped, paused, resumed, and looped. The only reason why them, by themselves, are not sufficient, is that the sounds can not be stacked. For sounds that need to overlap on top of each other (like the shooting sound), the SoundEffects class was needed. SoundEffects are very simplistic, all that can be done with them is telling them to play. Their strength comes from the fact that they can overlap other sound effects. For example, the primary weapon plays a single shot sound whenever it fires, but because it fires multiple times per second, the sounds overlap, making it sound like a machine gun.

2 Testing

2.1 Procedures

Initial testing was performed through a paper prototype (See figure 2.1.a). Due to the limitations of a paper prototype, no real-time gameplay elements were tested. The focus was on the shifting and exploration elements (See figure 2.1.b/c), which occur respectively in atomic and semi-real time.

The first testing session with the paper prototype was conducted internally, with all three developers involved.

Soon after the internal testing was completed, external playtesters from CISC 226 were briefed on the game features being tested and were asked to use them to get to the goal. Developers performed game responses to the player's actions in a manual "wizard of oz" style. Playtesters were left to accomplish the goal, with developers only providing guidance where they became stuck. Such instances were noted. Once the playtester had accomplished the task on several randomised maps, they were directly asked about the experience with the prototype.

More playtesting was performed once a partially functioning digital prototype had been created. A presentation of the game to the Queen's Game Developers Club preceded some feedback by the members assembled. Comments from the presentation as well as the informal playtesting were recorded. Friends and neighbours were asked to play to get a sense of difficulty and how the controls feel.

Final informal playtesting at the end of year show provided feedback from a wide audience. Feedback from this event was also recorded.

2.2 Results

Sessions as described in section 2.1.

Internal testing with the paper prototype confirmed that the non-real-time gameplay elements operated as we expected when designing them. We identified several contrasting routes of implementation and mechanic details. Some of these are: exactly how the tiles should shift, how the contents of shifting tiles should react, how to effectively balance the desire to simply shift yourself to the goal (or the goal to you), how tiles that are being pushed into the board should be generated, and what should happen to tiles and contents that get pushed off the board.

External testing with the paper prototype did not identify any major problems with the shifting or exploration mechanics. It was realised that scaling the size of the levels could be an important

factor in learning the tested game mechanics, particularly that a small level would be easier to learn on than a large one.

External testing with the early functional prototype earned praise for the game as it existed. Some players identified the relation to the game Labyrinth, which is an inspiration behind the tile shifting mechanics.

At the end of year show, UI problems with the were highlighted in the shifting interface, particularly that the queueing of shifting actions did not follow established HCI design. The addition of a more structured tutorial sequence would most likely effectively familiarise players with the new interaction method. Additionally, a UI bug was highlighted that compounded confusion experienced, and it has since been fixed.

Interestingly, an element of emergent gameplay first identified in internal playtesting of the digital implementation was spontaneously found by more than one player at the end of year show. When a player exits the tile grid through an external door at the edge of the map, they are damaged at a rate that allows them to renter another tile without instant death. At high enough skill levels, the player can quickly bridge one or even two tiles that have open external doors, allowing them to avoid making an additional shift, especially when collecting the final objective. This initially started out as a mistake (setting the out-of-bounds damage too low), but we liked how it introduced a “risk vs. reward” mechanic so it was left in the final version.

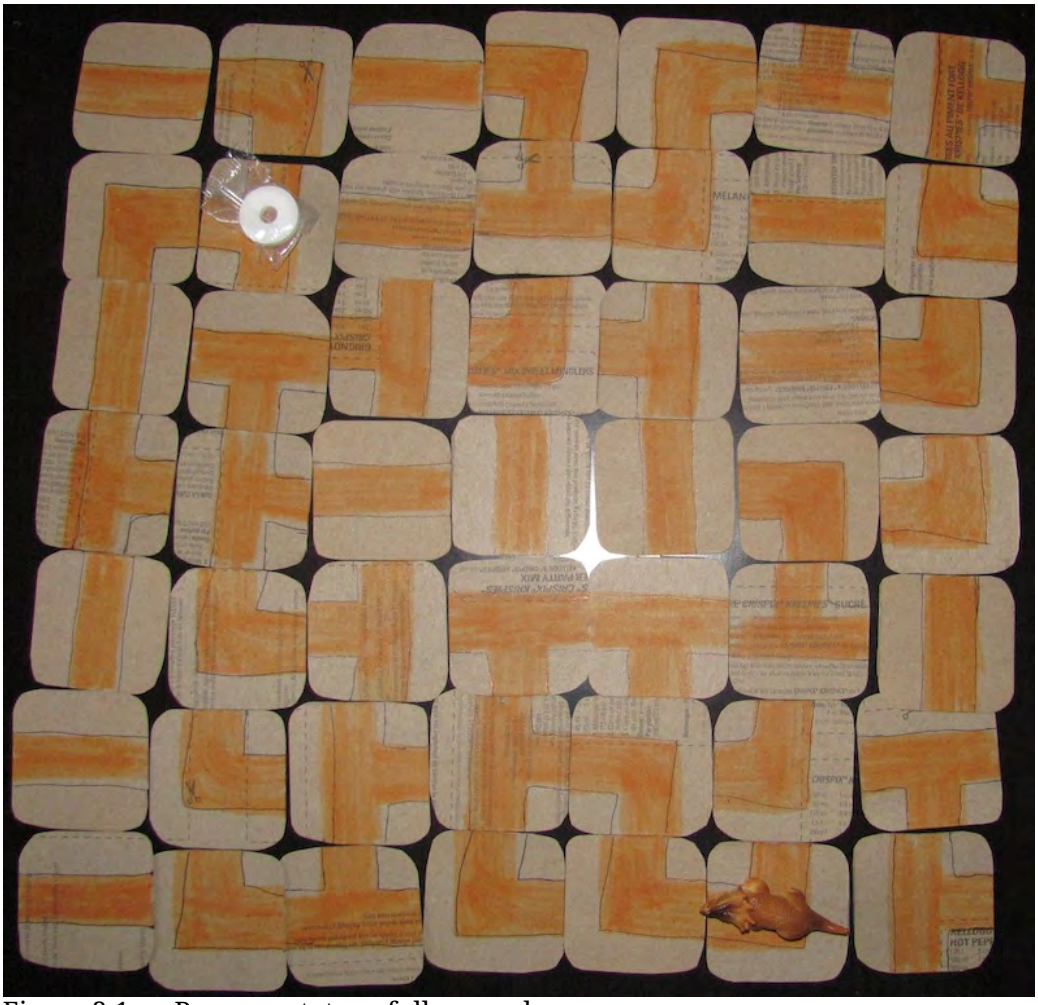
Figure 2.1.a - Paper prototype full example map

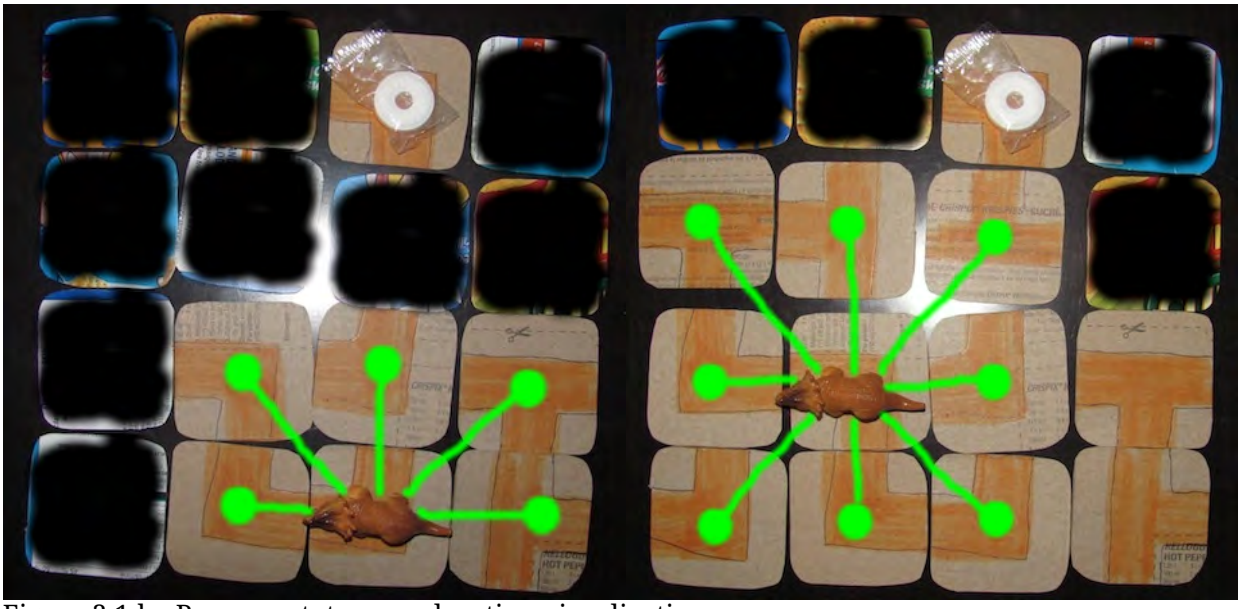
Figure 2.1.b - Paper prototype exploration visualization

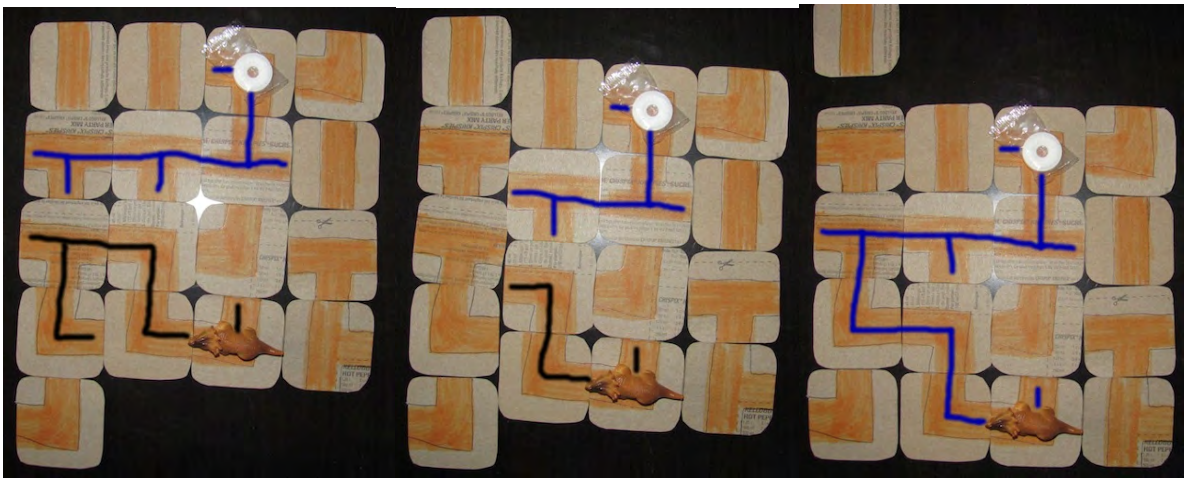
Figure 2.1.c - Paper prototype shifting visualization

3 Lessons Learned

3.1 Start small

We started with the initial idea of collecting objects in a maze that the player could shift around. Everything else was just fleshing out that one game mechanic. Even just implementing the basic game that we demoed at the end of term show was a lot of work. Had we started with a more complex idea, or tried to add more features to it, this project would have been way out of the scope of what's possible in a semester, especially one filled with other courses.

3.2 Use version control software

We used a Mecurial source code repository privately hosted on bitbucket for all of the code and assets associated with our project. Words cannot describe how essential this was to the

development of our project. Without version control software and never having to worry about who had the latest version of the code, sending code by email, manually merging code together, and other annoyances that other groups encountered, we never would have finished this project.

3.3 Test early

Get people to test your interfaces and gameplay. Things aren't always as intuitive to other people as they are to you. There is also the issue of you, as the designer, getting extremely good at your game and, as a result, making the game way too hard for new players. Make sure to get feedback on your game early, before you end up solidifying code and having to completely restructure things later.

3.4 Plan before you code, but don't be afraid to refactor

Design a solid foundation/framework before writing a single line of code. This initial design probably won't stick, but it will be a lot closer to the final architecture than if you just sit down and start coding. With that being said, don't be afraid to refactor your code as you go. It makes both adding extra features and altering existing ones a ton easier.

Do not start by implementing a gameplay slice (e.g. just the combat in a game like this). It makes adding extra layers of complexity (levels, etc.) more difficult. It's better to start with a simple version of everything, so that right from the beginning you can expand on all fronts. (This in moderation, since it's easy to get into too much if you start on everything at once)

References

Fullerton, T., C. Swain, and S. Hoffman. *Game design workshop, a playcentric approach to creating innovative games*. Morgan Kaufmann, 2008. Print.