

Phase 4 Report

Andy Lu, Andy Cheng, Martin Lau, Jackson Nguyen

Description

Maze Caller is a 2-D maze crawler where the player is trapped inside a maze. The player's job is to obtain collectables which increases their score, while progressing through the maze. They must collect the ladder before being able to proceed to the next level. While doing this, enemies are in the way and are chasing you. The game also features some light combat elements where the player can attack an enemy if they have collected a sword.

Changes in Plan and Design

In general, we were somewhat faithful to the original design outlined in the Phase 1 UML diagram. We still have a tree of classes inheriting from a single Entity class, and we maintained a class that contained a 2D array of Entity objects and an array of Entities.

The elements for the Menu are not present, however, instead replaced by making use of the Java Swing library. Some classes in the Entity inheritance tree are also missing as they were found to be redundant.

Because our original design was underspecified in many areas (for example, there is no mention of taking user input anywhere), while we might follow parts of the initial setup, the current design covers a much larger set of functionality and so the original design became less relevant as development progressed.

Lessons Learned

We learned a bit more about the real scope of making larger projects, like games. It was a much bigger undertaking than we had thought, and there was much more that we needed to take into account when developing the project. In the future, we should spend much more time hashing out a proper design and researching development of other similar projects. This did come to bite us in the back later on during development. Changes are inevitable, but a creaky foundation certainly doesn't help.

Documenting one's work was also an important lesson, as now our code would be seen by other people. Adding to the functionality was difficult without adequate explanation and understanding. More descriptive variable names, method docs, and inline comments for specific portions of code are all things we need to work on as they will make it easier for those working on it to understand what is going on in the code.

Proper organization and use of Git was something we had to get used to. Originally we developed a branch for each member, but this created integration issues if a member's branch

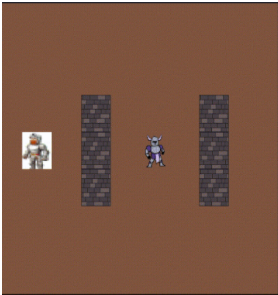
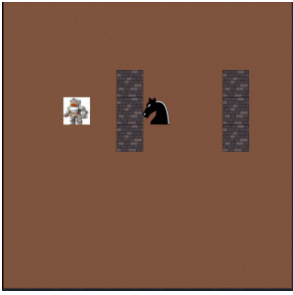
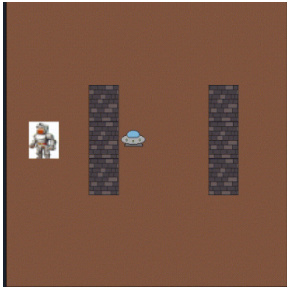
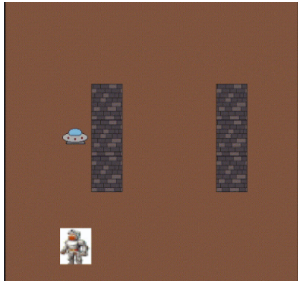
was significantly changed. We moved to developing a branch for each feature, and found it much smoother. Making more use of Git’s features, such as merge requests, might have also eased development and made meetings shorter.

Something that we did well on was having meetings many times a week. This helped speed the process of the development of the game as everyone would be up to date on what was going on and what they were working on. Questions would be answered about the development of the game. We would know what needed to be done next. Doing this allowed everyone to be on the same page.





Tutorial





<div><p>INSTRUCTIONS:</p><p>You are an adventurer trapped in a dungeon And you need to find a way out! Use keys to unlock doors, and ladders to escape out of holes You will also have to dodge or fight enemies</p><p>CONTROLS:</p><p>Up/left/down/right arrow keys to move up/left/down/right Spacebar to attack if you collected a sword R to restart if you died (hearts <= 0 or score < 0)</p></div>	Here we have our instruction screen, which tells the player how to control the character. The controls implemented are using the arrow keys, up/left/down/right respectively with spacebar to attack when a sword was collected and R to restart the game.
---	--

Next we have multiple enemies, the knight, the horse and the patrol

The Knight	The Horse	The Patrol (does not detect player)	The Patrol (detecting player)
			
The knight moves towards the player at ½ speed throughout the game.	The horse moves towards the player at ¼ speed in a L shape direction, similar to chess.	The patrol moves towards the player when it's within their line of sight. If not in line of sight, patrol simply moves left-to-right.	The patrol changes colour from green when they're not detecting the player, to red, when the player has been detected.

Next we have all the features that our game holds.

Door	Hole	Wall	Spike Trap
			
Here we have a door which can be unlocked with the key.	Here we have a hole which a ladder is needed to head to the next level.	Here we have a wall which is placed around the map to block the players movements.	Here we have a spike trap which deals damage to the player.

Keys	Sword	Coins	Ladder
			
Here we have a key which is used to unlock the door.	Here we have a sword which the player can obtain and use and attack enemies.	Here we have the coin which will multiply the current score by 2x.	Here we have a ladder which is used to be able to head to the next level.

JAR/Javadocs Files

They are located under /game/executable and /game/docs for JAR and javadocs respectively.

You can use `java -cp executable/project276-1.0-SNAPSHOT.jar dev.project276.main.Main` to run the executable, assuming you are in the /game directory.