Brief Overview

Throughout the development of the project, we realized the true importance of proper planning and design. A lot of the time throughout the course of the creation of our games we had to think on the spot and our approach to creating the game was somewhat chaotic. When we first created our UML diagram in phase 1, I believe we did a good job at making it (as things looked very organized and modularized). However, during phase 2 our group felt very pressured to finish the game and as a result, we ended up completely ignoring our UML design from phase 2 our game code looks very unstructured (with low cohesion and high coupling) and was simply just spaghetti code. Later during phase 3, our group ended up doing a lot of refactoring in order to effectively test different classes and methods. Testing or phase 3 was the most difficult phase for us as a lot of modifications needed to be done to the game code prior to us creating our unit and integration tests. We strived to have high coverage and test everything we could but the way our game was structured made it very difficult. We tried our best to follow the original plan and design as many of the things related to the UML and our use cases were useful and implemented. Our original plan was missing a lot of different classes like our KeyManager and MouseManager classes and I believe these things were missing from our UML because of how inexperienced we were with any sort of game development. We tried our best to keep our UML updated with all the new classes, methods, and relationships and to stay organized with our product backlog and sprint backlogs. With our development, we decided to take an Agile style of development, more specifically Scrum. As a group, we planned to consistently meet for daily 15 scrum meetings to discuss different aspects of our game. As noted in the phase 2 document, we have laid out the different tasks for each sprint. The product backlog we created back in the planning and design phases of our game helped our group stay organized with what we needed to implement next for the game.

Final Product Variations

With our implementation, a lot of our files were not packaged and were disorganized. In phases 2 & 3 we moved our classes to the packages entity, main, and menu. This allowed for better organization of our files and classes. With our implementation, we had many different problems with our hierarchy. With our original design, we did not really implement any sort of interfaces or superclasses for our classes. For instance, with our different types of menus, we decided to create a menu superclass that is shared between all different menu classes. Furthermore, with the inmate and guard classes, we decided to implement a MovingActor superclass that both the inmate and guard classes can inherit. Additionally, fixing these hierarchies allowed us to remove a lot of the duplicate code that existed within each of these subclasses. One of the biggest challenges we encountered with our code was the creation of the guard movement and tracking. This was difficult and constantly subject to change throughout each of our phases. We finally decided to have the guard follow the inmate only when the inmate gets in close range of the guard. With our map designs, we decided to add many more objects to the game to make it look more visually appealing. The process of adding more game entities became a lot easier after the refactoring we did in phase 3 to the EntityManager class and its setUp method (see phase3report in documents). Additionally to finalize everything we made sure to constantly test our game and obviously play the game repeatedly in order to find any bugs with our game. Prior to this current phase, many of our fields lacked any sort of information hiding or use of access modifiers. As a result, we decided to add these access modifiers and create the getters and setters associated with them.

Important Lessons Learned

With our design, we learned the hard way. In the beginning, we started off strong with a decent design but after phase 2 we began to ignore the plan we laid out from the start in order to have everything finished on time. We believe that if we were to spend more time on the design and planning phases that a lot of the mistakes and problems we encountered, later on, could have easily been avoided. The problem we encountered, later on, was with maintenance and modification. For instance, when we wanted to create new levels this process became very problematic as before all game entities were hardcoded into the game. This is an example where refactoring truly helped us save both a lot of time and effort in terms of extendability. As a group, we learned the importance of constantly refactoring code and following good design principles. Furthermore, the idea of modularizing and breaking long pieces of code down helped us debug our code and help us locate where problems were with our game. Additionally, we how useful Object-Oriented Design can be with extendability and any future maintenance we had with our game. Java is a very Object-Oriented language and when we first began working on the project we were not taking advantage of everything it had to offer. Concepts like inheritance, polymorphism, abstractions, and generalizations were very advantageous to us when we began using them as they allowed for our classes to follow better hierarchies and less duplicate code, take for instance the Menu and MovingActor superclasses that we implemented (look at phase3report in Documents directory for more details). With that being said we believe the biggest takeaway from all this was how important communication and assertiveness were in working in a collaborative environment. One of the biggest challenges for all of us was balancing the workload for this course with all of our other courses. Over time we learned to work around this and have other group members step and be accountable for completing some of the work. Communicating with one another and having that sense of openness allowed us to make it this far with our game and have fun making it along the way. Overall our group dynamic was very good and we all put in equal amounts of work.

Tutorial

Entity Descriptions and functions

Background

In Prison Escape players are faced with the goal of escaping Arkham prison. Filled with guards and traps, players are forced to collect all the keys scattered around the map and escape. Players will be put to the test, through 3 different levels (increasing in difficulty).

Level Timer

Players are forced to escape the prison within 100 seconds. If the players fail to do so they lose the game. The level timer is displayed at the top of the game.

Level Score

Players begin the game with 0 score points and they lose whenever the score is below 0. The level score is displayed at the top of the game.

Inmate (Main Character)

The main character of the game consists of a prison inmate (dressed in white and black jail clothes). You the player play as the inmate and can control the inmate using the WASD keys.

Guard (Enemy)

The main enemy of the game is the guard that chases after the inmate when in close proximity. If the inmate is not in range the guard will continue its regular movements (ricocheting from wall to wall moving back and forth). If you listen closely you can hear the heartbeats of the inmate when in range of the guard.

Escape Door (Maze Exit)

The escape door is how the player will escape the prison. Players must collect all the keys in each of the respected levels in order for the escape door to open. Level 1 requires 3 keys, level 2 requires 4 keys, and level 5 requires 5 keys.

Keys (Escape Door & Score collectible)

Keys are the main collectibles in the game and are needed in order for the player to win the level. Once more, the number of keys depends on the current level the player is in. Furthermore, collecting all the keys give the player a chance to exit through the escape door.

Chicken Drumstick (Score collectible)

Chicken Drumsticks are placed randomly on the map and start flashing away before they disappear and reappear again. The chicken drumstick grants the user a 100-point increase to the game score.

Clock (Timer collectible)

The clocks that are placed around the map grant the players more time to escape from the prison. Collecting the clock adds 10 seconds to the game timer.

Laser Beams (Trap)

A very dangerous trap, the laser beams are placed around the map by the guards to help catch you. Avoid them at all costs. Colliding with the laser beams decreases 50 points from the game score.

Guide for first Game:

1. When the game is first launched players are faced with the main menu
2. Press on the New Game button or press Continue to play from the last time you played the game
3. After creating a new game inmate is spawned inside level 1. Players must use WASD to control the inmate and can press escape at any moment in the game to bring up the in-game menu
4. The player must collect all the keys avoiding all guards and traps
5. Once all keys have been collected the player can escape through the escape gate located on the middle right side of the map. The gate will also start flashing to indicate that the player now has access.
6. Complete all 3 rounds to finish the game

Game Functionalities

<u>Guard Chases after inmate</u>
When in close proximity the guard begins chasing after the inmate

<u>Inmates' heartbeats when in range of the guard</u>
Which in range of the guard you can hear heartbeat sounds until the inmate gets out of range

<u>The guard becomes “Dumb” when the inmate is out of range</u>
The guard moves back and forth and ricochets with the walls it collides with until the inmate gets back within the range

<u>Inmate death sound and game lose sound</u>
When the inmate dies you can hear him screaming and then you can hear the losing sound

<u>Inmate collecting Key</u>
Adds 1 to the total level key count (shown on the top of the game) and makes a key sound on a collision

<u>Inmate collecting Chicken</u>
Adds 100 points to the level score (shown at the top of the game). The chicken is at a certain location on the map for only a small period of time. When it is about to disappear it will start flashing. Colliding with the chicken makes an eating sound.

<u>Inmate collects Clock</u>
Adds 10 seconds to the level timer (shown on the top of the game). Colliding with the Clock makes a clock ticking sound.

<u>The escape Door starts flashing when all keys are collected</u>
The Escape Door starts flashing after all keys have been collected.

<u>The game ends when the score is less than 0</u>

<u>The game ends when time runs out</u>

<u>The game can be saved after exiting the game before</u>
Given the player has started a new game before and has left the game, they can load the game from where they left off by pressing the Continue button in the Main Menu.

<u>Players have the freedom to leave whenever they want</u>
Within the game, players can press ESC on their keyboards to bring the in-game menu and they can then press the Back to Main Menu button and then Quit the game.

<u>Players can view game credits in main menu and after 3rd (final) level is complete</u>
In the main menu, players can click on the credits button to view credits and also if players complete the 3rd level they can view the game credits after pressing the next level button in the Win Menu

<u>Players can get instructions on how to play the game</u>
In the main menu players can click on the Help button to see the controls and entity descriptions