# Gold Girl

CS302 Java Game Report

Kenny Li & James Flood

# Table of Contents



# Table of Figures

# 1.0 Introduction

We have been tasked to develop an offline Java game that is inspired by the 1980's game Pacman for a twelve-year-old girl. The aim of the game is for Pacman to consume as many pellets as possible while avoiding collision with the ghosts.

We have developed our game using Java 9 and JavaFX and designed it similar to the original game but altering features such as different enemies, win conditions and power-ups while having a storyline to cater for the client.

# 2.0 Minimum Requirements

The minimum requirements that were set by the client are:

1. A welcome screen with the option to select single and local multiplayer.
2. Stationary characters and a countdown timer before the game starts.
3. Allow player characters to be controlled by the keyboard while having the AI characters move automatically.
4. Have a windows size between 1024 x 768 and 1440 x 990.
5. Allow player characters to consume pellets upon collision, with an increase in score.
6. Restricts the player character to move through walls unless there is a gap on the side of the wall which allows the character to wrap-around through the sides of the window.
7. When the characters collide, there must be an appropriate notification.
8. The game must have a two-minute limit per level. Pressing the 'PgDn' key should skip the time to 0.
9. The game must be able to be paused and resumed by pressing the 'p' key and be able to exit to the main screen by pressing the 'Esc' key.
10. A win condition must be evaluated then have an exit screen with a summary, that allows the user to return to the main menu.
11. There must be appropriate sounds played throughout the game depending on actions and collisions.

# 3.0 Base Features

## 3.1 Welcome screen with multiple game modes

We have designed our menu using Scene Builder with a window size of 1024 x 768 pixels. The user has the choice to either use keyboard inputs or mouse inputs to navigate in the menu. From the menu, there is the option to play, set options, see credits and exit the game.

When the play button is pressed, the user will be able to have the ability to choose to play either single or multi player and pick whether they would want to play story mode or select a specific map to play on.

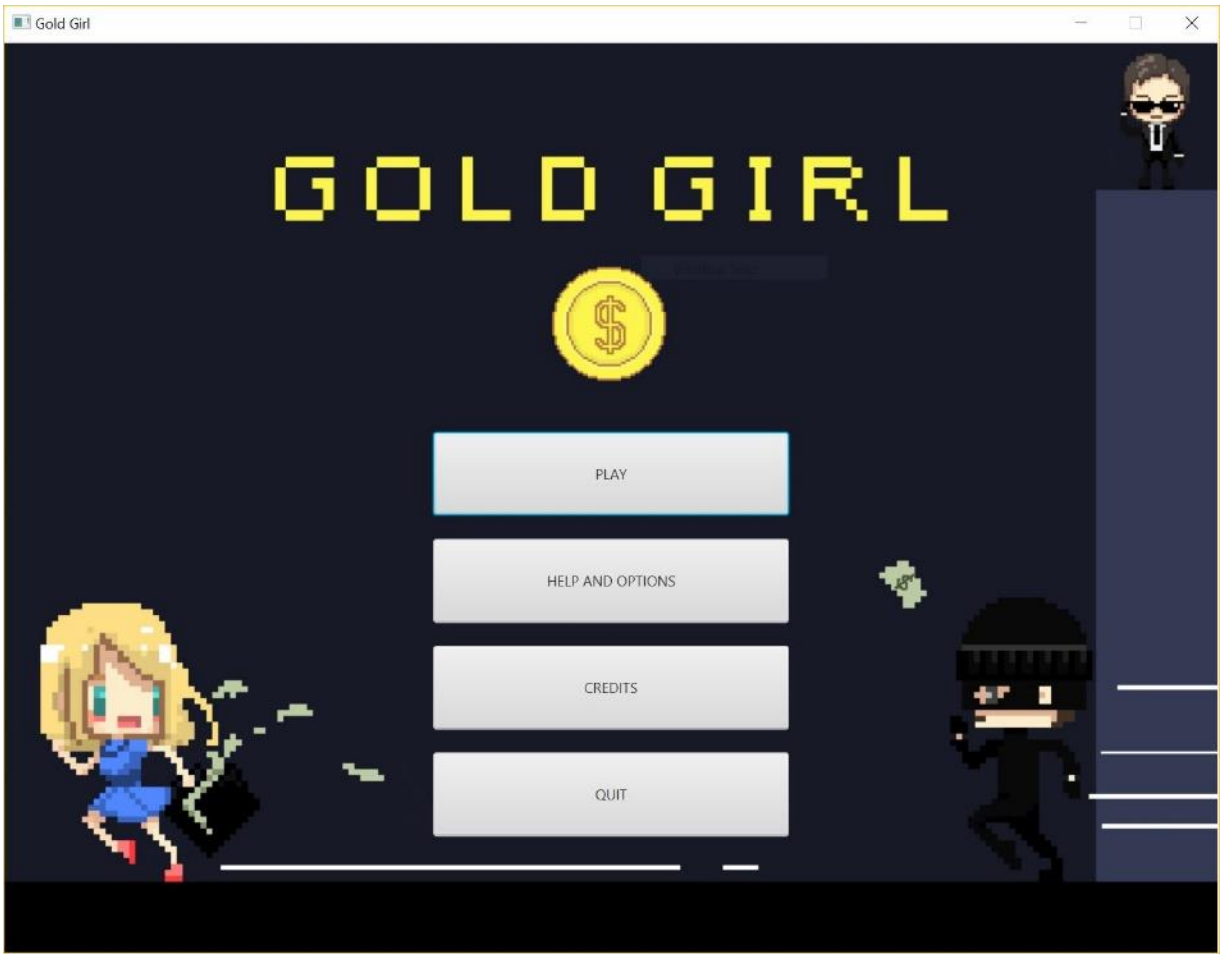


*Figure 1: This shows the Main Menu of the game*

# 3.2 Gameplay

When the game starts, the map, player and enemies are loaded onto the window. A countdown timer will start before any characters are allowed to move. After the timer reaches 0, the game time is set to two minutes and all the AI characters will start moving and the user will be able to move Gold Girl.

The movement of Gold Girl is restricted by the walls on the map. When Gold Girl collides with coins, they are consumed, playing a sound while increasing the score. When Gold Girl collides with an AI character, the score will decrease by a percentage while playing a sound to indicate when the collision occurs. The AI character will also slow down for a set time.

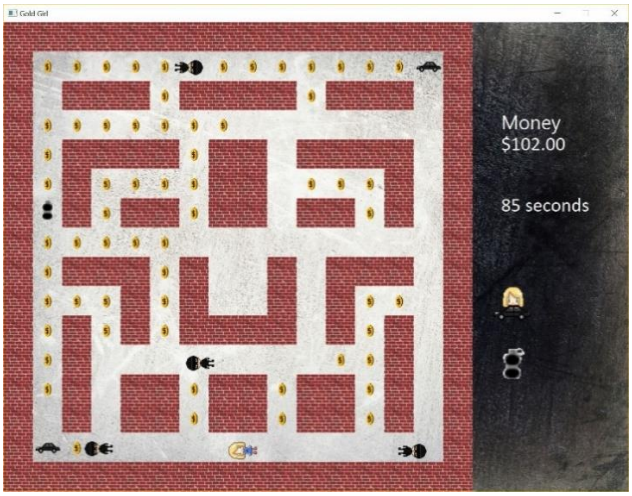


*Figure 2: This shows a screenshot taken midgame during the first level*

# 3.3 Game win conditions

There are multiple ways to trigger the game to win. The user can either collect all the coins spread on the map or play until the game timer runs out. Once the game ends, an exit screen will appear, providing the score and giving the user the ability to return to the main menu by pressing the 'Enter' key.

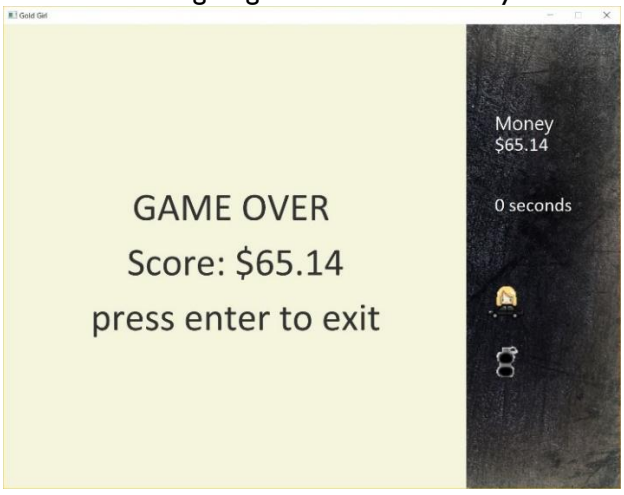


*Figure 3: This shows the game after the character has completed the game*

## 3.4 User Control Keys

The user is able to pause, exit and skip to a time of 0 seconds by pressing certain keys. Pause is assigned to the ‘p’ key and exit is assigned to the ‘Esc’ key with a prompt to confirm whether the user wants to exit or not. The key ‘PgDn’ is assigned to skip the time to 0 seconds while ‘PgUp’ is assigned to skip to the next level.

# 4.0 Additional Features

Additional features were added to enhance the gameplay for Gold Girl. They were also added to provide a more modern and entertaining approach to the Pacman inspired game.

## 4.1 Audio and Visuals

A smooth jazzy theme was chosen to play while the game is running. There has also been an options menu implemented to control whether the music and sound effects are turned on or off. The sound effects that occur during gameplay represent the items or characters that Gold Girl collides. Each item also has its own unique sound effect.

## 4.2 Items and Power-ups

Multiple new items and power-ups were also added to alter the actions of Gold Girl. These include pellets that provide a different score for each level. They are represented as coins, small and larch piles of cash, gold and bitcoin (in the bonus level).

Power up items were also added that activate special abilities for Gold Girl. The flashbang will stun all enemies on the map for five seconds and not allow them to interact with Gold Girl. The other power-up is a car, which when collided with Gold Girl, increases her speed and allows her the ability to collide with an enemy and stun them without losing any score for ten seconds. These items are limited so strategic use of them is crucial to getting the highest score possible and making it through the story levels.

## 4.3 Levels

Different levels were added to enhance the experience of the game. There is a total of 4 levels, each one having a different map and theme. The first level is an easier level used as something similar to a tutorial in order to familiarise the user with the controls and behaviour of the game. The second and third level introduces a new enemy, causing the game to become more difficult, requiring more thought into where you move.

There is also a bonus level not included in the story mode where the player must collect bitcoin. Each bitcoin has random value to represent the volatile nature of some cryptocurrencies.

*Figure 4: This shows the different maps for each level*

## 4.4 Story

When the story mode is selected, an initial storyline is presented to allow the user to immerse into the game more. The story bases of the progression of levels, changing themes from the neighbourhood, to the city and finally the gold mine.

## 4.5 Enemies (AI)

There are two different types of enemies with their own specific behaviours. The Robber characters are more numerous and fast. They chase the player down and steal a percentage of the players cash when they catch her. The Agent character is slower but ends the game when he catches the player.

In multiplayer, there is the choice to have up to two player-controlled Robbers. The rest of the Robbers are subsequently controlled by AI.

# 5.0 Design

## 5.1 Suitability of Java and Object-Oriented Programming

Java as a language was a good fit for our game, due to its use of multi-threading and its object-oriented nature. The multi-threading allowed us to implement features such as the multiplayer mode with ease, without worrying about reading the multiple user inputs with polling or appropriate methods. As such we were able to focus primarily on the design of the game and less so on the 'nuts and bolts' of a game engine.

As our design had elements which were repeated throughout the game, OOP allowed us to design classes and use them throughout the design. For example, we designed a basic parent character class which had many generic methods that the game engine would use regardless of the character type, then we implemented child or sub classes that inherited all of the parent's methods and contain more specific

methods or fields that only the specific character type used. Another example is the coins that the player collects through the levels. The ability to repeatedly use objects instead of having to code every instance made the process much easier.

## 5.2 Software Development Strategy

Our group did not use a specific development strategy, but we did use some elements of organized strategies. We used an iterative and incremental approach where we introduced features and game elements one at a time. This was effective for us as we had little experience using Java and allowed us to try things out and have them work, which allowed us to then plan the next logical step and divide the workload out. This approach also allowed us to test things out and find bugs and glitches during the development process, so we could modify our design.

We added features primarily to the GameController class, to make sure that they worked as intended, then moved them later to separate classes. This was important to ensure that our design was not coupled too closely and made each class internally cohesive.

## 5.3 Class Structure

When launched, the game loads the MenuController, which allows the user to select the game mode and when chosen, a GameController is loaded and is responsible for interacting with all classes that the game needs. To maximise cohesion internally within classes and minimize coupling we designed our code to interact primarily with these two controllers, which called upon the other required classes accordingly.

## 5.4 Development Issues and Possible Improvements

Originally, we had intended to use a pathfinding algorithm such as Dijkstra's shortest path, or A* to have the enemies chase the player. Unfortunately, as we were quite far in the development process, we found it difficult to have our existing code work with these algorithms. We instead went for a compromise that used the relative distance to the player.

If we had much more time it may improve the design to add in a system that uses the previously mentioned algorithms by having the game use a grid system rather than building the maps based on pixel distance. This would allow us to use a more accurate collision detection system, where the current version works 90% of the time, it does lead to some control issues when it fails.

Another improvement would be to include a high score system, possibly even with online scores so that the user can compete with other players around the world.

If the control scheme could be adapted, the game could potentially be ported to android so that it could be played on mobile devices.

Fleshing out the story further would help the user enjoy the story more and be more likely to keep playing. More scenes in between levels could aid in this. Finally, MVC structure could have been implemented better by planning specific classes ahead of time so that features would be more modular and less coupled. Removing some features in certain classes such as GameController and putting them into separate classes would also reduce coupling and make the code more modular. This would help to add more features easier in future. It is also good practice as it would allow others to contribute to the game without having to look through multiple classes to find out how existing features are implemented.

# Appendices

## Appendix A: Class Diagram