Amal Chandra & Kunal Bhatia

(Mind Benders)

Abstract

Super Princess is a PacMan inspired game, developed using Java

COMPSYS 302 REPORT

Super Princess

**Aim of the Project**

The aim of the project is to create a game like Pacman using Java programming language that would be suitable for the client’s 12-year-old daughter. The overall goal is to make a Java application which can take inputs from the user and have the system respond accordingly. We were provided with a set of minimum requirements in our project brief which were necessary, and we need to develop the game with our own twist to the original Pacman game.

**Introduction to the Game**

The game, Super Princess, is a Pacman like game. The aim of the game is to collect all the diamonds and, in the process, try to kill all the knights. There are powerups/power downs available in the game which the princess can use to make the game easier/harder to finish. The princess will be placed in the maze and she has to go around the maze to collect all diamonds and try to avoid the knights unless she has collected the sword to kill them. The princess cannot go over walls and she has 2 minutes to finish the level.

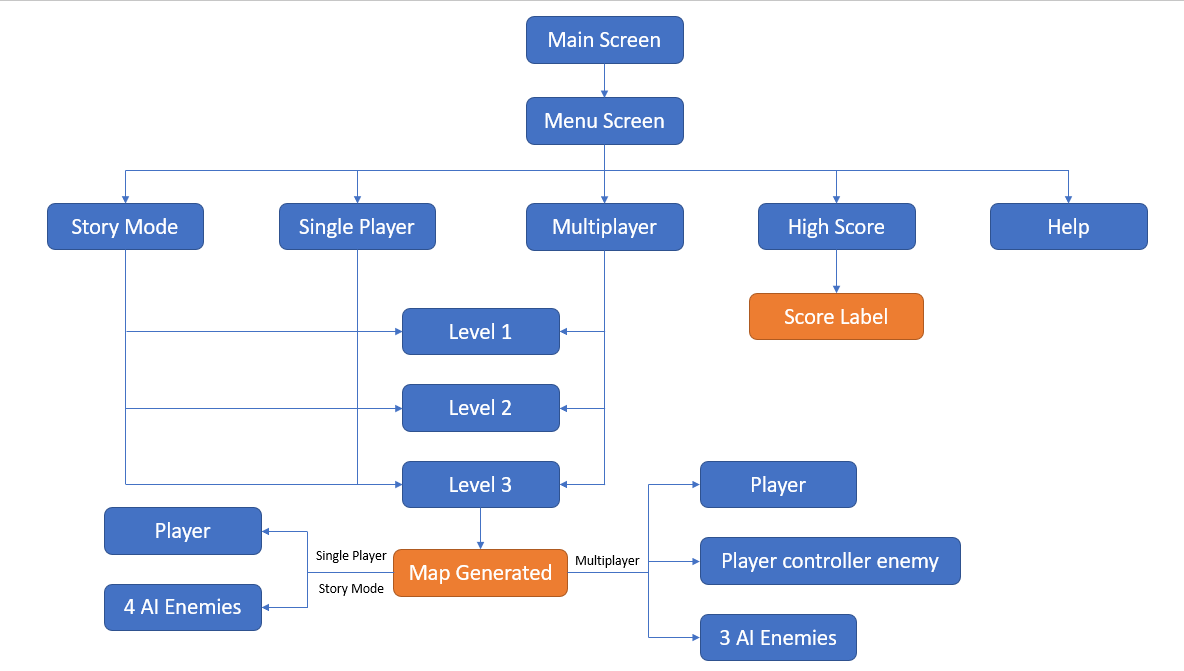
**Minimum Requirements**

The project brief provided us with the minimum requirements from the client. The minimum requirements are the following:

* A welcome screen should be presented, allowing the user to select a game mode using the keyboard. The minimum valid game modes should be single player (one character controlled by a human, four ghosts controlled by AI), and local multiplayer (one character controlled by a human, up to two ghosts controlled by humans, the others controlled by AI).
* When the game is started a countdown timer (from 3) should indicate to the user when the game starts. Before then, players should not be able to move.
* When the starting countdown finishes, AI-controlled ghosts should begin moving. The player-controlled characters should move according to the relevant keyboard events.
* The window should be at least 1024x768, and no larger than 1440x900 (unless the window is resizable).
* When the protagonist collides with a pellet, the pellet should be consumed, and the score should be increased.
* When a player-controlled character collides with a wall, the character should not be able to move through the wall.
* When the protagonist collides with a ghost, some notification should be shown to the user, and the protagonist should lose a life. If the protagonist still has lives, they should respawn, and the characters should reset positions, but consumed pellets should not reappear.
* The game should take a maximum of two minutes (per level) and show the time remaining on the screen.
* During normal gameplay, the ‘p’ key should pause and resume the game. ‘Esc’ (Escape), should be able to exit the game (a confirmation screen may be appropriate).
* If the protagonist consumes all the pellets without running out of lives, then the game should respond appropriately (either ending the game or moving onto the next level).
* Appropriate sounds should be played throughout the game depending on actions and collisions.

The minimum specification for the project that was set by the client were met for the game.

**A top-level view of how the system works**



**Issues/Challenges**

A major issue we had during the development of the game was with the AI for the ghosts of the Pacman game. This is because we both had very little knowledge about Java, hence finding the best/efficient way to code the ghosts so that they can go towards the user became a challenge. None of us knew any algorithm that we could use to code the ghosts. It is also because the development of AI was not as straight forward as other functions. We overcame this issue by looking at YouTube tutorials on shortest path algorithm , BFS and DFS and also by asking the Teaching assistants for help. The other issue we had during the development of the AI was that the implementation of code was hard for us to do therefore we again watched different YouTube videos to get help.

The other major issue we had during our development of the game was trying to understand what the other group member wrote. This is mainly because of the lack of comments we had during the development of the game, which made it harder for the other group member to understand what the code was. This also made it harder for the person to keep on track in terms of what was going on in the code as they would forget what they were doing. Because of this, the member would sometimes change some code that the other group member wrote, or the group member would create another function which was already implemented therefore wasting time. We overcame this issue by messaging each other and commenting on code or meet up over the break when we didn’t understand and tried to comment the code afterwards.

**Features that Improve Functionality of the System**

To improve the functionality of the system we implemented the following additional features.

**Power Ups / Power Downs –** The game character can pick up power ups such as speed, which will increase the movement of the player, slow down, which will decrease the movement of the player and sword which the character can use to kill the enemies. This powerups will be generated randomly when the game starts.

**Improved AI –** There are 2 different types of AI in the game, one is random and the other is the one which will try to follow the game character.

**High Score –** This is too keep track of all the scores. This makes it interesting for the user as they would always try to beat high score to be on the top of the high score table.

**Story Mode –** We have implemented story mode so that the player can get more engaged in the game. As having a story makes it more exciting than just going through different levels.

**Random Player Spawn –** In the beginning of the game the main character will spawn at random location.

**Discussion of the suitability of the tools for the application (e.g. Java, Git)**

The tools that we used for the development of the game were Java, JavaFx (graphics library), Eclipse and BitBucket.

The programming language we used to develop the game in was in Java. We used JavaFx which is a library of Java for the development of the graphics of the game. Java is a high-level language which made it easier for us to program as it was easier to understand as it is more in terms of human language than machine language. It also has a lot of built in functions which meant we could use those functions rather than writing all the functions down. Since Java is object-based programming that means we only had to make the object once and could use it multiple times. Which saves us time from writing repeated code. Java is a very popular language therefore we could easily find instructions online if we got stuck on something. The disadvantage of Java that we experienced is that it uses a lot of RAM while running the game. The other issue we experienced while using Java was that it was not as easy to drag and drop like it is to do using Scene Builder. We tried using Scene Builder, but it was hard to understand and modify the code manually.

We used BitBucket to store our code and for us to easily upload code so that another group member can access it. BitBucket also allows to store multiple backups of the code and if we make any mistake when programming we can easily get back to the point where the code is working. BitBucket was also useful for merging the different codes from the different group members which removed the hassle of copying code to one file for different members.

**Discussion on OO design and how cohesion and coupling issues were addressed**

Object Oriented Design allows the code to be easy to understand as it is based on objects rather than actions and data rather than logic. This makes it easier to separate all the objects into different classes and writing functions for those objects in their respective classes. This also helps in avoiding writing the same code multiple times. Having object-oriented design future changes to the code is easier as changing it in the objects class will change it everywhere it is used. It also enables to reuse the same code for different projects if necessary.

Cohesion and coupling issues and easy maintainability of the code were addressed by using Model View Controller (MVC) pattern. This separates the code, so that all the classes which were used to show to the user were put in View. While Controller was used to make changes to the Model. We created different packages called Model, View and Control, and the files were sorted appropriately, which made it easy to separate all the files. We want high coupling and low cohesion in our code. We achieved high cohesion. This is so that we can make changes to one part of the code and so that it doesn’t affect the other parts of the code.

For us using the Model View Pattern and Object-Oriented Programming made it easier for us to design the game.

**Discussion of the software development methodology (e.g. IID)**

The software development methodology that we used for developing the game was Iterative and Incremental Development. We used this method for the game development as it was easy for us to understand and because it matched with our style of software development. In this process we would develop different parts of the game at different times, as we tried to program the easier codes first and then move on to the hard code. While we were doing this, we would go back to the code and improve what we did before. Once we wrote the code, we would test it straight away to see if the code it is working or not. We also used test-driven methodology, this is where we would test if the code works for certain requirements and then work onto improving it to meet more requirements. This method was used for the part of the code which we found difficult to program.

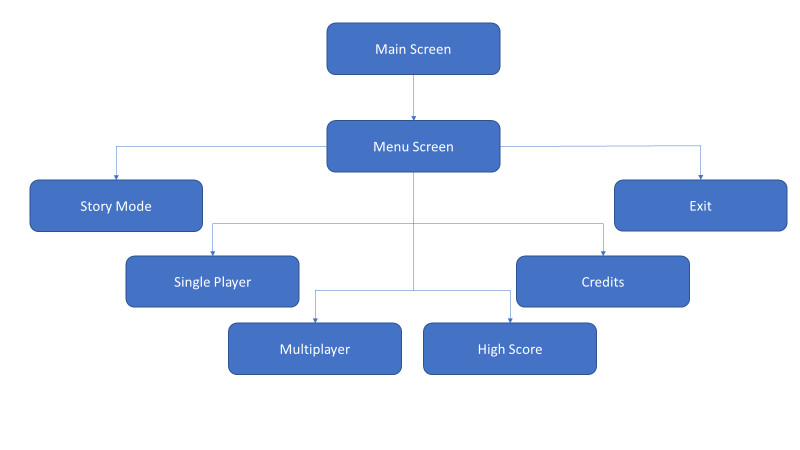
**Suggested Improvements for Future Development**

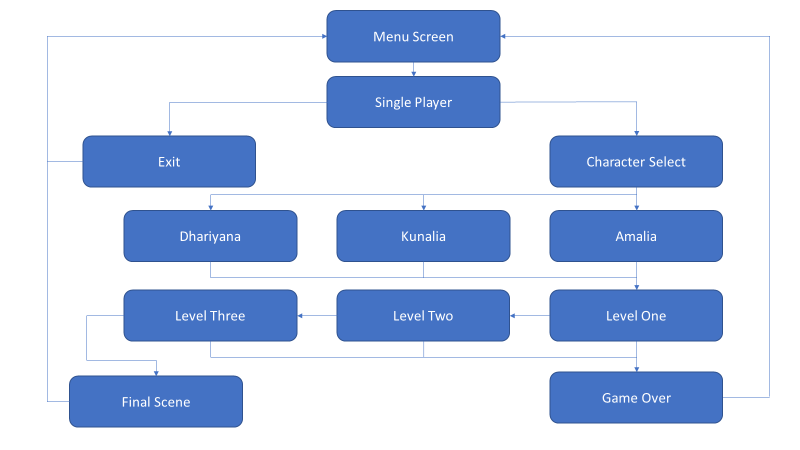
The improvements for future development for the game would be to add more features and follow the MVC pattern more closely to get high cohesion and low coupling. The extra features that would be added to the game are:

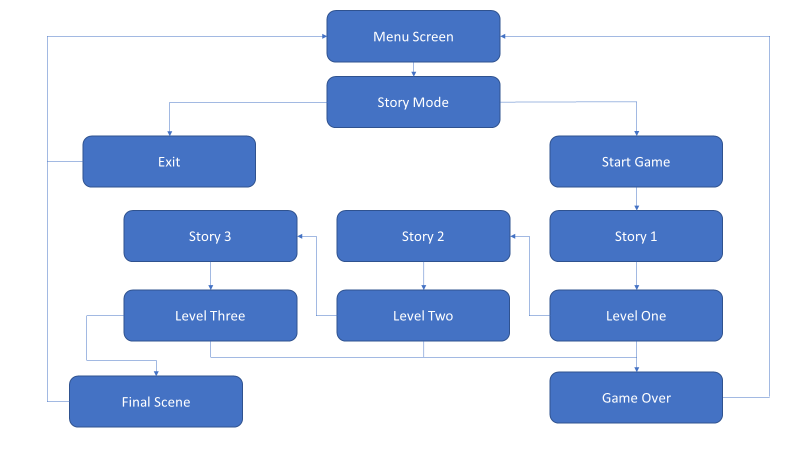
* Random Generated Map – this would allow each time the user plays the game it would always be unique.
* More modes
  + Endless Mode – With the random generated map this would allow the user to keep on playing the game until the user loses all their lives.
  + VS Mode – In this mode there will be 2 players versing each other. The player who gets the most coins and kills the most enemies wins the game.
* More characters – more characters with different skills would make the game more unique as the user can choose the character depending on which skill their want.
* AI – improved AI and with different enemies having different characteristics.
* Learn Java better which will help in making Java applications.

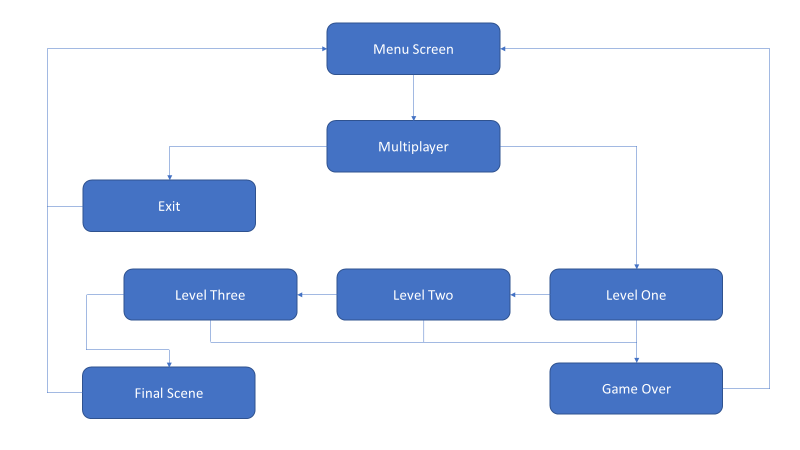
**Appendix­­­­­­­­­**

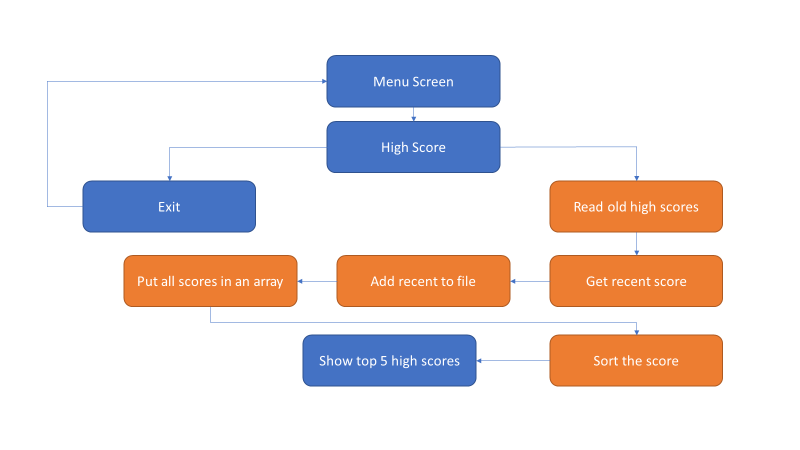
**Map for Menu Screen**

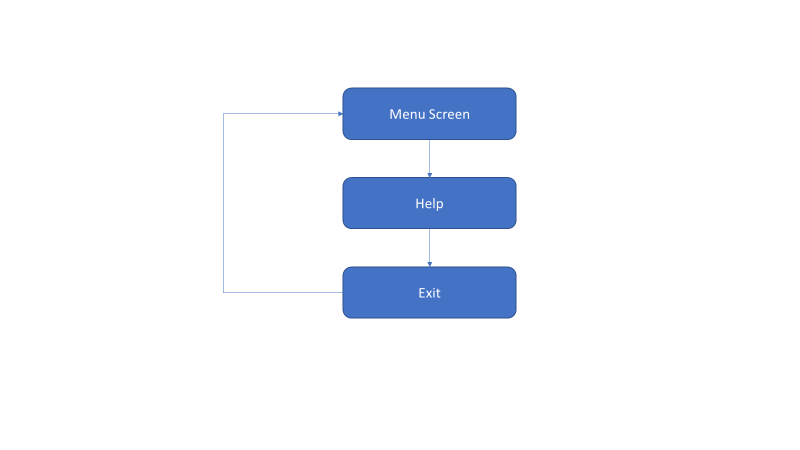
****

**Map for Single Player**

**Map for Story Mode**

**Map for Multiplayer**

**Map for High Score**

**Map for Help**