A revised version of pacman created using Java. Run on the Linux OS.

Pacamana Jane

COMPSYS 302 Assignment 1

Henry Shen, Lincoln Choy

Project Overview

What were our design features?

For this game we decided parody Indiana Jones, so our pacman character has become a female explorer called Pacamana Jane. The story involves venturing to different themed areas and collecting all the artefacts (pellets) all while escaping from the ghostly pursuers. Instead of the regular powerup where pacman can eat the ghosts, our character is given a whip to attack the ghosts at a range.

For the A.I of the ghosts, we used Dijkstra's shortest path algorithm to find the best path to a location on the map. Each A.I had a slightly different behaviour.

Type 1: Random movement, at each node it would randomly pick a direction to move in.

Type 2: Chases Pacamana Jane. At each node it would recalculate the shortest path to Pacamana Jane and pick a new direction based off that information.

Type 3 : Chases Pacamana Jane until it gets too close, then it will start moving to its specificed corner until there is enough distance between them. At each node it would recalculate the shortest path to Pacamana Jane or the corner and pick a new direction based off that information.

Type 4 : Chases Pacamana Jane until it gets too close, then it will start moving randomly until there is enough distance between them. At each node, if it far away from Pacamana Jane, it would recalculate the shortest path to Pacamana Jane and pick a new direction based off that information.

We also added a gas trap, which spawns in a random location on the map. This happens every 15 seconds and any character that contacts it will die.

All our image assets were created by us using MS Paint, Gimp, and Photoshop. We tried to give it an overall temple/ruin feel and so all the images were influenced by that.

How were the requirements met?

Our game welcomes the player with the Main menu screen, where you can scroll through the 5 buttons using the arrow keys and select a button using the enter key. There is a play button which will bring you to mode select and map select. A tutorial button which shows a series of slides which walk through the controls and mechanics of the game. A leaderboard button to display the top 3 high scores. A credits button to display the credits and references. Lastly, an exit button to close the application. The game allows up to 3 players to play the game, where 1 person controls Pacamana Jane, and the other 2 will be controlling a ghost. The remaining ghosts are A.I controlled.

The game starts with every character frozen in place, unable to move for 3 seconds. Once the countdown ends, the A.I controlled ghosts begin to move and the player characters will move depending on the keys pressed. You will also be able to pause the game with the p key, and exit the game with the esc key. Movement is restricted by the map walls, so the moving characters will stop when they hit a wall. When Pacamana Jane contacts a pellet, it is removed from the screen and your score is increased by 10. But when pacman collides with a ghost, a death animation is played for pacman. Pacman will lose a life and after losing all 3 lives, the game will end and the score will be recorded. If Pacman still has lives left, everyone will be reset to their starting positions and the game will continue from there. To beat the level, you must collect all the pellets on the map, and you only have 2 minutes to do so. Failing to complete the level in 2 minutes will end the game; you will be able to keep track of your remaining time on the top right of the screen. After the game has ended, a results screen will appear and tally up your score. You will then be able to select whether you want to play again or exit to the menu.

The game window is set to a size of 1366 x 768.

Sound effects on:

- Menu scrolling
- Menu selecting
- Eating a Pellet
- Using Whip
- Pause
- Beating a high score

Features which improved the functionality of the system:

Many public setters and getters were used, which improved the encapsulation of our code. We had a controller for each separate view, which lowered the coupling of the system. Although this is standard for MVC design patterns, it improved/helped when debugging. We also used inheritance via composition and extension for reusability of classes (mainly for GameObjects). We created a public interface which would be implemented by moving game objects only, and this interface was named MovingCharacter. This would allow us to differentiate between character objects and stationary objects. We put our moving objects in an array list to update AND draw these objects, while putting the other objects in a GameObject array list, and only draw those on each frame.

Issues Encountered and our Design Methodology

Significant issues which arose and how they were overcome:

Character Movement:

Our character movement went through many changes over the course of development, and was probably the most time consuming part of the project. The initial implementation used hitbox collision to detect where the walls were. When a character collided with a wall, its position was reset to the point right next to the wall. This was usable, but only if the characters were player controlled as the A.I could not detect when to turn. We then switched to a grid/tile system, where the map is represented by a 2D array. The paths that the characters can take are also defined in the map. This way, we could easily navigate the map without hitbox collision, and a method based on x,y position comparison allowed for a naïve implantation of the ghost A.I. But even with this set up, it was not good enough. The A.I would often get stuck under certain conditions, and it would not always be able to take the shortest path to Pacamana Jane. So, we had to find a way to calculate the shortest path from the ghosts current position to Pacamana Jane. The method we used was to create a Node class which reads through the map information for the turning points and creates a node for each one. The nodes are then checked for related nodes and that is stored in each node object. Now that we have a way to represent distances and paths we can use Dijkstra's algorithm to find the shortest path. This was used in conjunction with the 2nd version of movement to create our final version. To complete this, it took around 2 and a half weeks.

Management :

There was a lot of disagreement on certain features being implemented, or certain animations. This was usually resolved via asking for play testers' opinions, and ultimately which features could be more efficiently implemented into our design.

Why is Java suitable for this application:

Java is suitable as it is an object oriented language, using classes to differentiate objects a good way to design code. Also the JavaFX library is very good for creating applications which involve a lot of animation and sound playing. C++ also has a library called SDL which allows the user to create an interface for a game, but it does not seem to have support for xml layout files. Also C++ requires the designer to delete their own objects created on the heap. Java's automatic garbage collection system allows the user to forgo deleting their own objects created on the heap.

How we addressed OOP in the project:

We used OOP design to create different classes for different game objects(Pacman,Ghost,Pellets,Wall) , standard OOP design. If these objects were needed to create a different game, with the same characeters, we would be able to copy-paste and reuse the code for these game objects. We also used a Game class which would handle/look after all the objects created in one game session. A GameStateController class would be used to handle user input to the game and would interact with a composited GameViewController object which looks after the animation of all the objects in the Game object. All code was designed with maintainability and reusability in mind. We also used classes to differentiate views from each other.

Software Development Methodology:

Our software development methodology was an iterative and incremental type.

We could reuse a few classes from a previous project that we wrote together. So, we had less iterations to go through when developing it for this game. This was the case for classes like Pacman, Ghost, Board. We had a general idea of how to implement them so many aspects were easier to do.

However, for many newer classes it was necessary to go through lots of trial and error until it behaved the way we wanted. Even after there was a working version, often we would improve on the behaviour, so at all stages there was an iterative process.

Final Thoughts

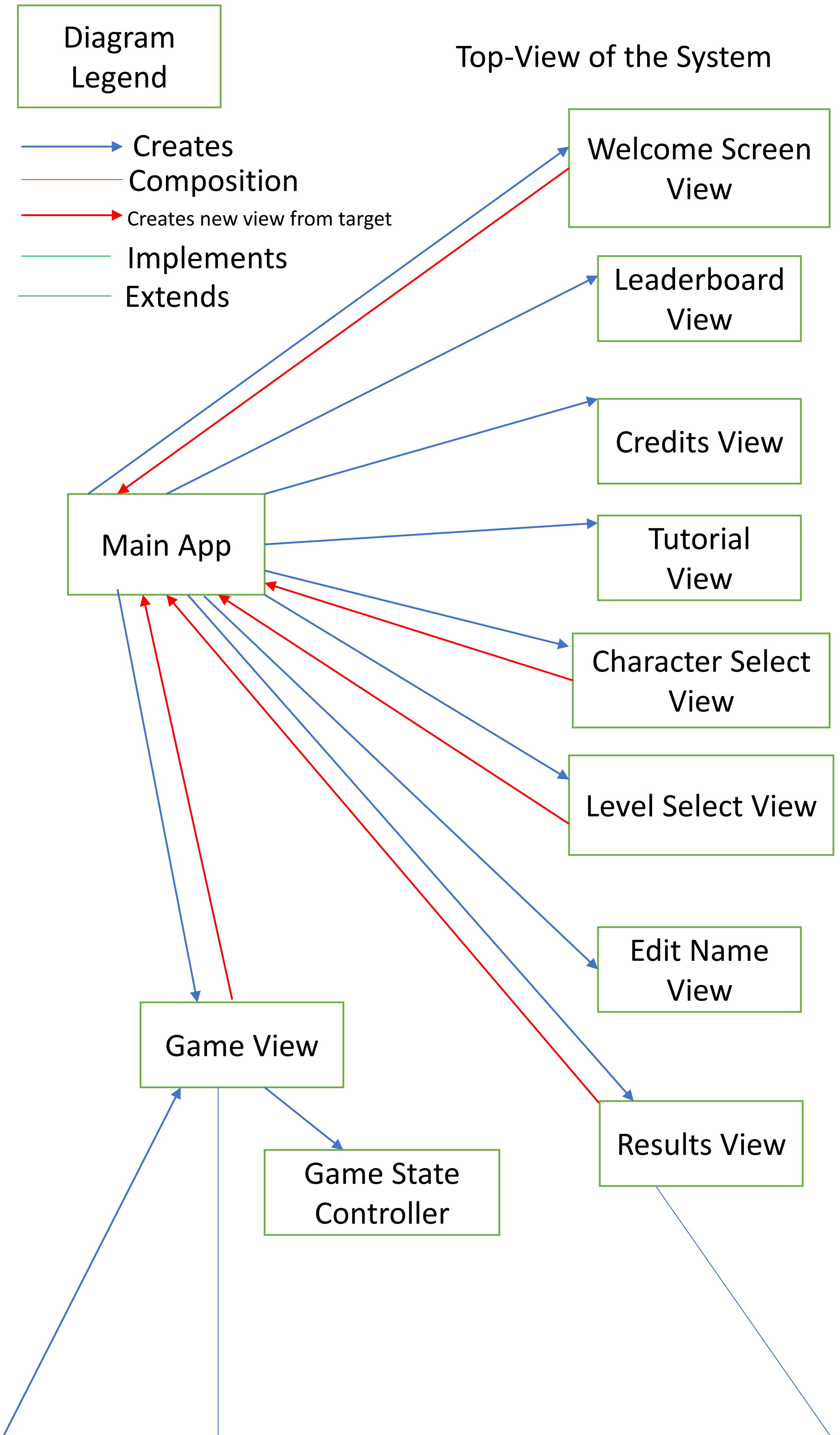
Discussion of the game design experience:

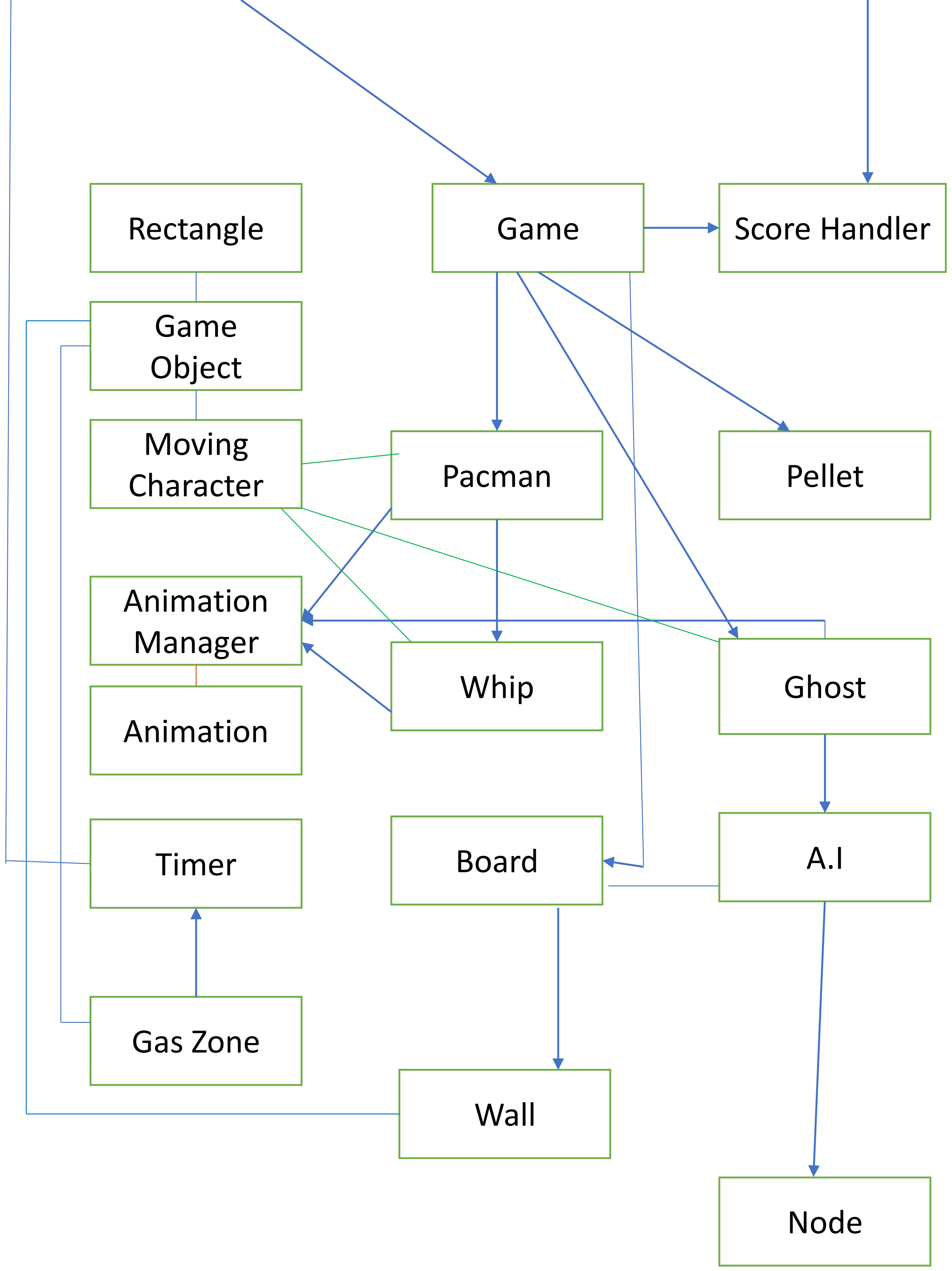
It was interesting to think of new ways to improve pacman, while still being true to the original game. There were many ideas that we had brainstormed, and narrowing it down to just a few was harder than we had first thought. Though in the end, we were not able to implement everything we had planned for, and a lot of features were tweaked during development. It really shows how things will not always go as planned, and through doing the project we can see that there is no way to create the perfect game on paper. Our current knowledge of java also prevented us from bringing some of our ideas to life.

Suggested improvements for future development:

The timetable that we set for ourselves at the beginning of development did not include all the things that we had to do. As we approached the due date, we realised that there were still quite a few things we had to add to the game for it to be considered “finished”. For example, recording high scores, pausing and exiting the game, playing sound effects, finding and fixing bugs in the game. Next time we should try to keep these things in mind and include them in the planning stage so we do not end up rushing near the end of the project.

Top-View of the System




Rectangle
Game
Score Handler
Game Object
Moving Character
Pacman
Pellet
Animation Manager
Whip
Ghost
Animation
Timer
Board
A.I
Gas Zone
Wall
Node