Rohan D’Souza rdso323 819388789

Amar Parkash Singh asin579 750905783

Phase-1 Report

# How the developed system meets the requirements

**Brief:**

The aim is to create a game like Pacman, one of the most popular games of its time. The brief was vague which allowed us to incorporate our own ideas and themes to the product. What we have come up with is a themed version of Pacman that incorporated styling and features from the anime series Dragon Ball. The aim of the game is for the protagonist (Pacman) to each as many pellets (Ramen) as possible to get a high score, whilst avoiding his enemies (Ghosts) to prevent losing lives. The goal was to produce the game using the programming language Java and its scripting platform javaFX.

When designing the application software, design principles are used to meet the minimum specifications, but also for the overall end user experience. The software design principles that were used in our application are graphical user interface(GUI), artificial intelligence(AI), object-oriented programming (OOP) and the layout of the code using MVC.

Our game (Pacman Dragon Edition) is based on the theme of the anime series Dragon Ball Super. The user plays the protagonist Goku who is required to eat all the ramen to gain strength and defeat his foes for the survival of his universe. Goku can collect dragon balls throughout the game to gain a temporary powerup to defeat his foes, however he must eat all the ramen to gain enough strength to protect his universe. The primary objective of the game is to get the highest score and protect the universe.

# Features that improve functionality of the system

**Artificial Intelligence:** We believe the AI is one of the key features of our game. With 3 different types implemented, it’s extremely difficult to evade the enemies all the time. To the layout of the map and the way in which the enemies are executed there is a sense of them being able to trap you in and leave you with no possible way to escape. The AI are also based on characters from the Dragon Ball Super.

AI was also one of the features that took the longest time to make due to the different types that had to be coded for.

**Soundtrack:** A custom sound track plays in the background of the application.

**Collectables:** Power ups that allow the user to defeat his/her enemies.

**High Score:** Saves top scores to a file so the user can see who the best is.

**Extra life:** An extra life the user gets an extra life if they survive through half the game.

**Story:** The game is based on a story, so the user is part of something bigger.

**Help screen:** The screen provides the user with the controls and a background story for the game.

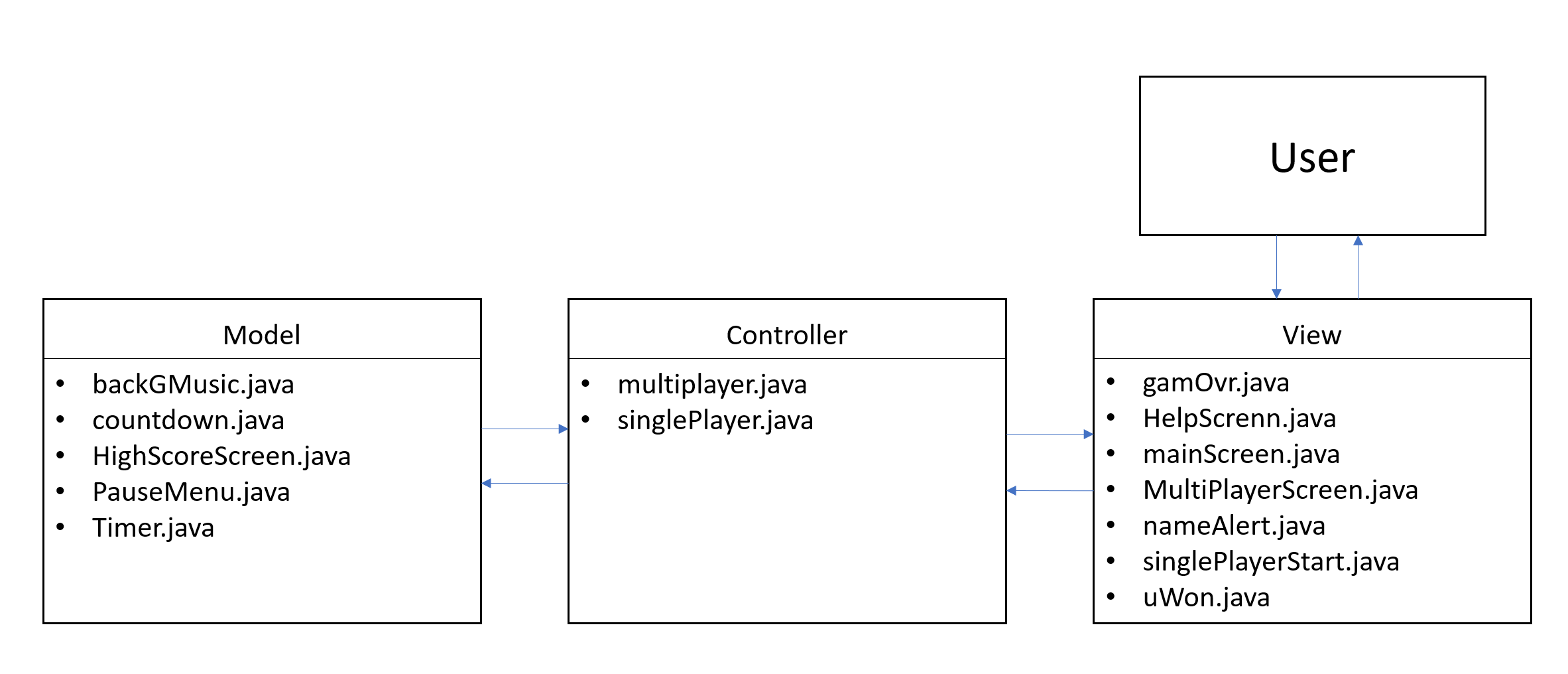
**Photo Editing:** The photos were edited so that they do not have white background in the game.

# A top-level view of how the system works

To manage the software development, we used the model-view-controller(MVC) design structure. MVC is a design arrangement to keep the model, view and controller data separate for better code maintenance.

MVC organises the code into three categories:

* Model: The model stores the game objects and data. The model objects retrieve and store data, such as the name of the user and the score achieved. The model classes in our project are:
  + High Score Screen: Retrieves the name of the player and their score, writes that information in a text file so it can be displayed. Also reads from the file and displays the data.
  + Timer: Manages the amount of time of in game play time the user has left and is then displayed to the user.
  + Countdown: Allows the user to get ready for the game using a 3 second countdown before the animation begins.
  + Pause Menu: Pauses the game and shows the user the options of resuming or exiting the game.
  + Background music: Creates the object music to be played when game starts.
* View: The view is the user interface which is the entry point for the application. It receives input from the user or users and should perform actions accordingly. The view classes in our project are:
  + Main Screen: Displays the main menu to so the user can decide what to do. Also has buttons which allow the user to start a game mode or switch to the help or high score screen.
  + Help screen: Displays the help screen.
  + Single player start: Shows the user the story to the single player game, the controls and requires the user to input their name before starting the game.
  + Multiplayer screen: Shows the user the story to the multiplayer game and their controls before starting the game.
  + You won: Displays you win when the user has successfully completed the game and saved the universe.
  + Game over: Displays game over when the user loses the game.
  + Name alert: Displays warning when the user tries to start the game without entering their name.
* Controller: The controller contains the game logic, game design and handles the user input events. The controller classes in our project are:
  + Single player: Controls the single player game mode, reacts to the users input and reacts accordingly, has the AI to control the ghosts, sends info back to the single player start class to be displayed.
  + Multiplayer: Controls the multiplayer game mode, reacts to the user’s inputs and reacts accordingly, has the AI to control the ghosts, sends info back to the multiplayer screen class to be displayed.



Top-Level Design of the game

# Significant issues & how they were overcome

Key issues that popped were AI getting stuck at random points in the map and the ability to move the user with a single key press oppose to having the button pressed down.

The first issue happened quite frequently and meant that it was easy to evade the enemies and win the game. The problem occurred when the AI trying to get to the user but had an obstacle in the way. In these situations, we decided the best alternative was to code in a random direction to move based on a logical assumption.

The second problem was tough as it was hard to find similar examples online. In the end it became clear that the best way to address the problem was to set the direction we wanted to go in “true” if the button was pressed and all the other directions were set to “false”. This ensured the user would only change direction if a new button was pressed.

The most difficult part of the game development that we faced was sticking to the MVC design plan to ensure our code had high cohesion and low coupling. This was due to the concept of model-view-controller being a relatively new to us. We were partially successful in splitting up game design and logic into components. There were parts of the software design which we managed to get low coupling and high cohesion. Overall the high cohesion and low coupling was a mixed success, this would be something that could be improved on in the future.

# Suitability of the tools for the application

The tools that we used for the project are Java, JavaFX, Bitbucket, Google Docs, Microsoft Office Word, PowerPoint and VirtualBox running Ubuntu.

Using Java and the scripting language JavaFX for the game was challenging at time due to it being a high-level software development language. Java and JavaFx in terms of our game development went well because it was straight forward understanding what each line of code was doing and because there were a fair number of online resources available for us to utilise. However, for some problems that we ran into there was nothing available online. Having access to a game engine such as Unity or the Unreal game engine would increase the development options.

The git service we used (Bitbucket) was extremely helpful as we could work on different parts of the code and them come together and merge our code. It also was great for storing backups of our code in case anything happened to be deleted.

Google Docs allowed us to collaborate on the documentation for the project and enabled us to access the documentation from any computer. This allowed us to work together while being at different places and not worry about losing our work.

Microsoft Office Word and PowerPoint allowed us to take the documentation that we had done on Google Docs and format it, to make it look better by adding different layouts.

VirtualBox running the operating system (OS) Ubuntu was used by us because the final game was to work on Ubuntu. This allowed us to develop the game on the same OS and ensure there were no problems during the final submission.

# Discussion on OO-design and how cohesion and coupling issues were addressed

Object oriented design makes the code easily readable due to it being split into classes which contain functions relative to that class. These functions then get called by the controller when it is required to be executed. This also allows for functions to be reused, an example of this from our code is the pause menu which is called by the 2 controllers we have in our game (single player and multiplayer). This helped from writing the same code again with slight changes, which supports the idea of encapsulation. The coder only needed to know that they needed to call the pause menu function and didn’t have to worry about the code colliding from single and the multiplayer game modes. By making different classes for different entities of controller, view and model it made the code have higher cohesion. To reduce coupling, we had multiple functions in the classes that were required by a function in that class to be called often.

# Software development methodology

We decided to primarily use the waterfall methodology for this project. We split of at the start to work on the GUI and gameplay separately and merged frequently to ensure our code worked with each other’s.

While developing the game, when a new feature or part of the game was implemented it was tested multiple times so that when we moved on from that feature it was insured to be working correctly and that there were no bugs. To test the code, we would run the game and try the possibilities ourselves. Then we moved on to developing the next feature and adding it to the game.

When testing the code and the implemented features we also got our class mates and other friends to test the game. Some of them were able to find glitches which helped us improve the features. Amar also sent the game to his 11-year-old cousin who enjoyed the game very much and helped us improve it with her feedback.

Certain principles of the agile software development process were also implemented. We went from 1 step to the next as the waterfall process suggests. However, we also went back and made sure the implementation of the next step did not impact the previous and kept everything up to date. We also added feature that were not in our design documentation (i.e. AI).

# Discussion of game design experience

Overall, we believe it was a very enjoyable but stressful experience. It seemed like an improbable task at first given the time constraints and the limited amount of knowledge we had, but once the basic concepts were understood it was easy to further build on the code we had. At the later stages there was less looking online for answers and more time spent thinking about how we can go about solving the problem ourselves.

At the later stages it became essential to try and perfect the game as much as possible. This was achieved by constantly tinkering with the code. A good example of this would be the lives system that was implemented. At first the lives were in a stationary position and were removed if the player died.

Later on, however, we decided for an extra feature that we will give the protagonist an extra life half way through the game. We then tinkered with the code to make sure lives could be added back in and worked as expected. We also made a change to ensure the lives were always cantered regardless of whether there were two or one extra lives left.

# Possible improvements for future development

* Use scene builder or another software to create a graphical user interface that is more aesthetically pleasing.
* Use a graphical overlay to make the game look better.
* Enhance the AI by possibly using Dijkstra’s algorithm to find the shortest possible to path to the user. This would make them even more effective than they are now and possibly a little more difficult to combat.
* Add more levels if possible that could possibly relate to earlier versions of the Dragon Ball series.
* Use a more in-depth design could also help increase cohesion and reduce coupling.

# Appendices

**Screenshots from the Game:**

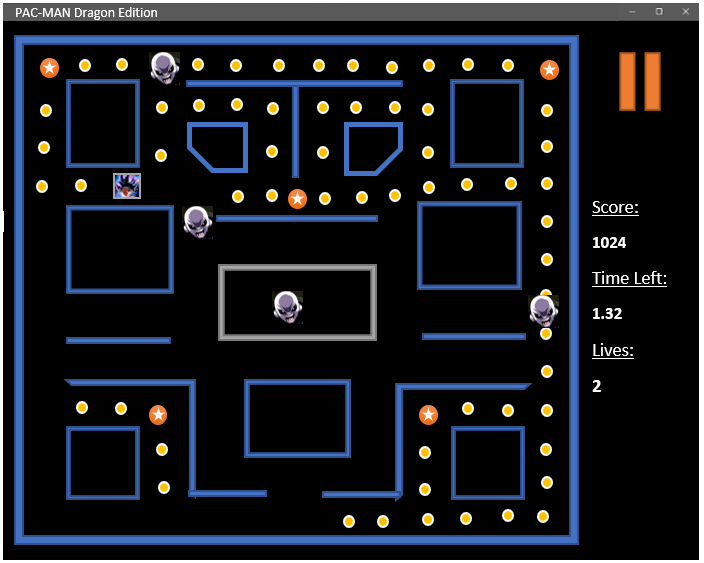
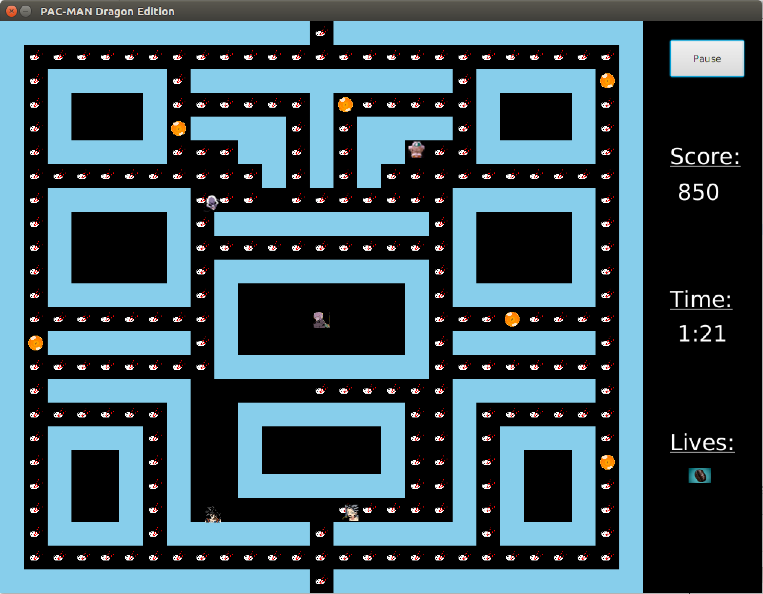
****

**Final Main Screen**

**Concept Main Screen**

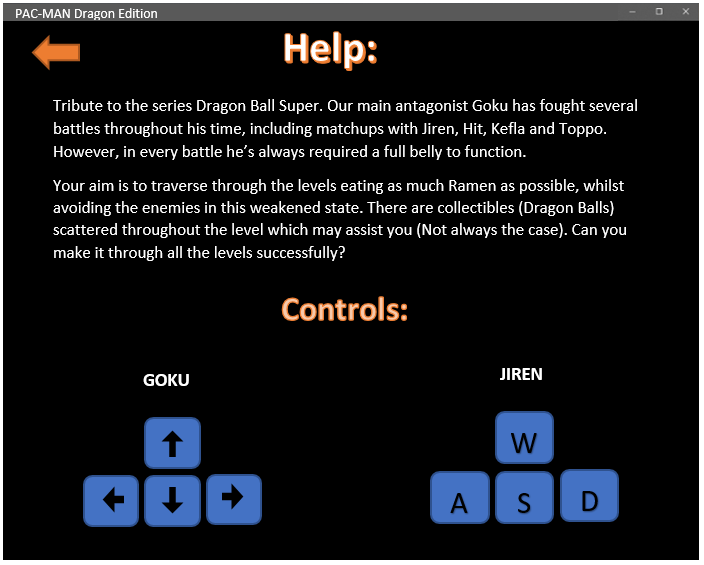
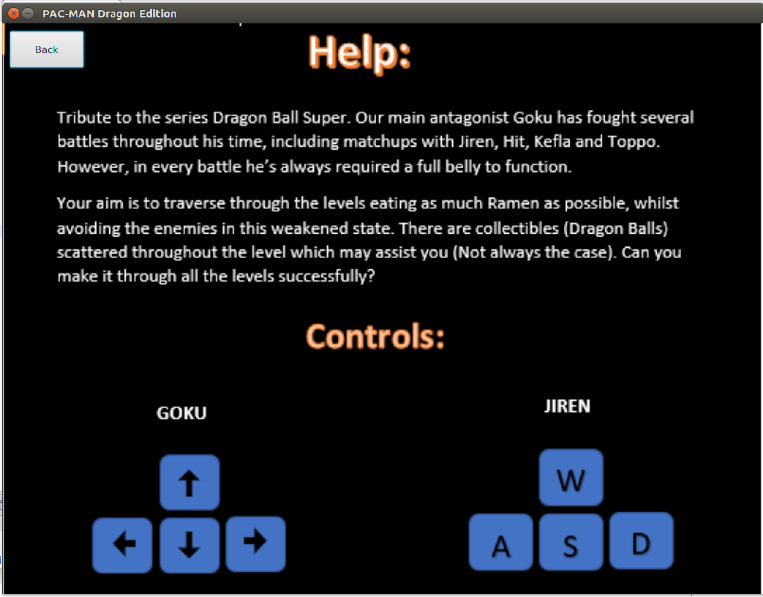
**Final Game Screen**

**Concept Game Screen**

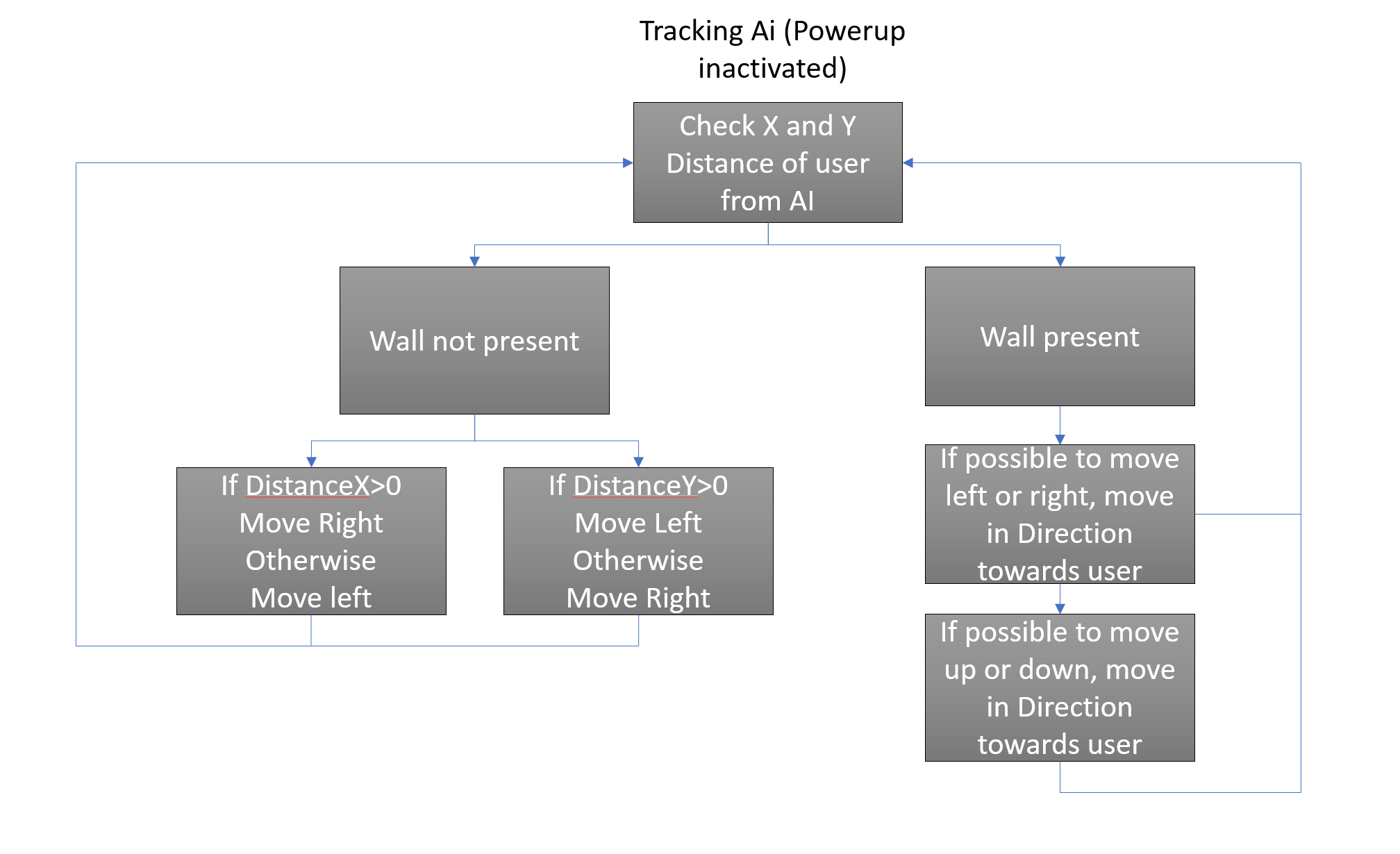


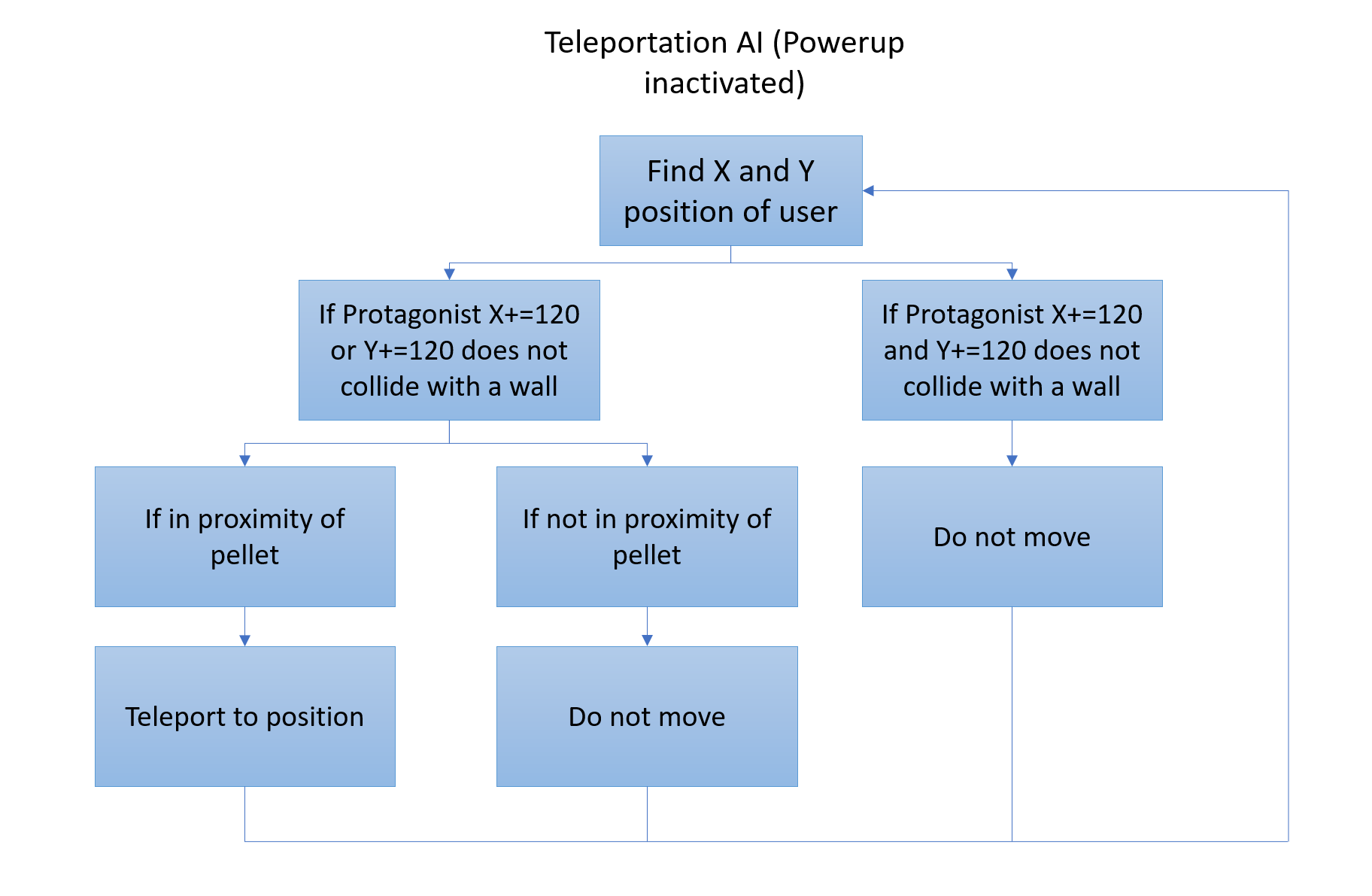
**Final Help Screen**

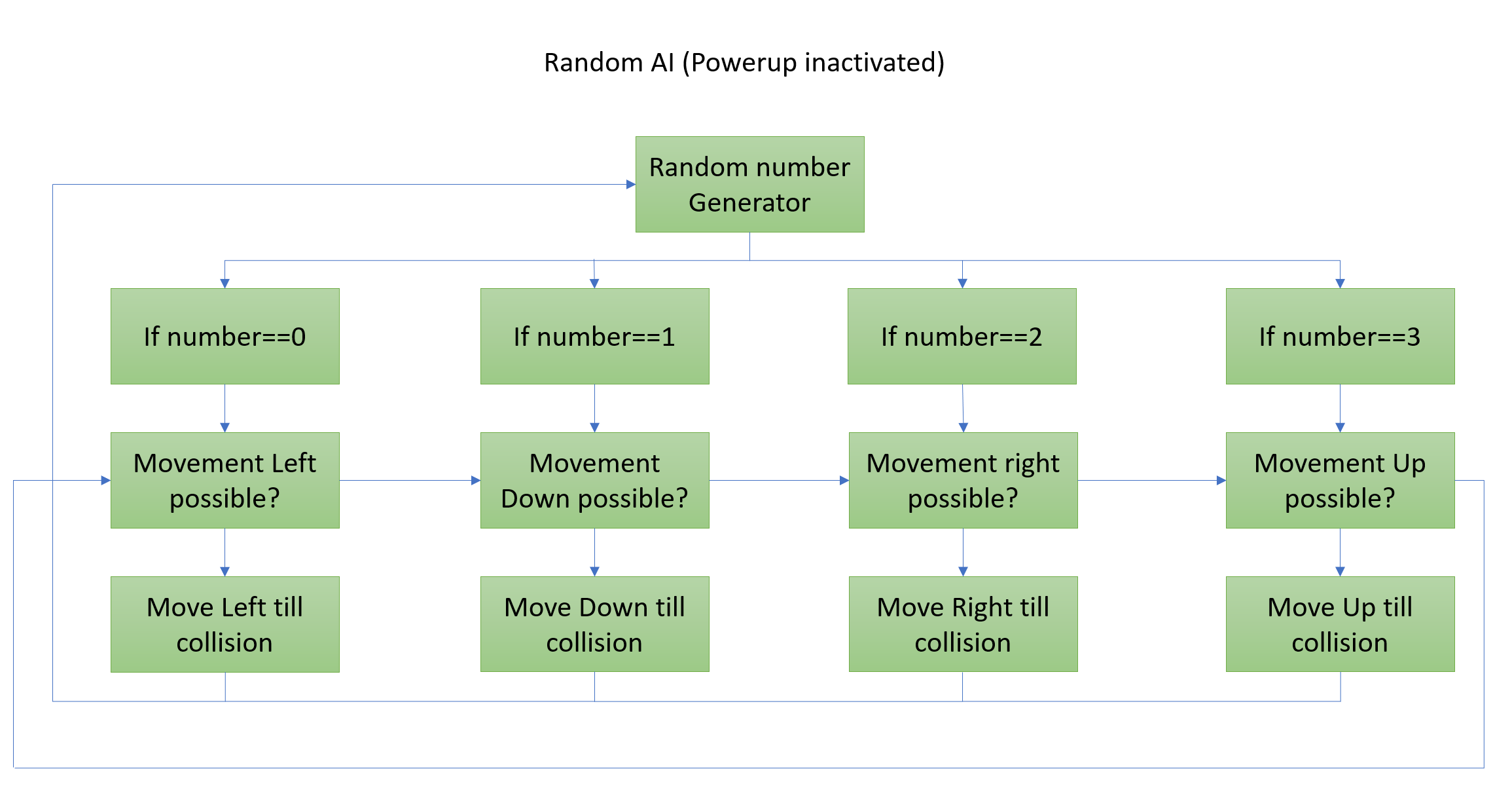
**Concept Help Screen**

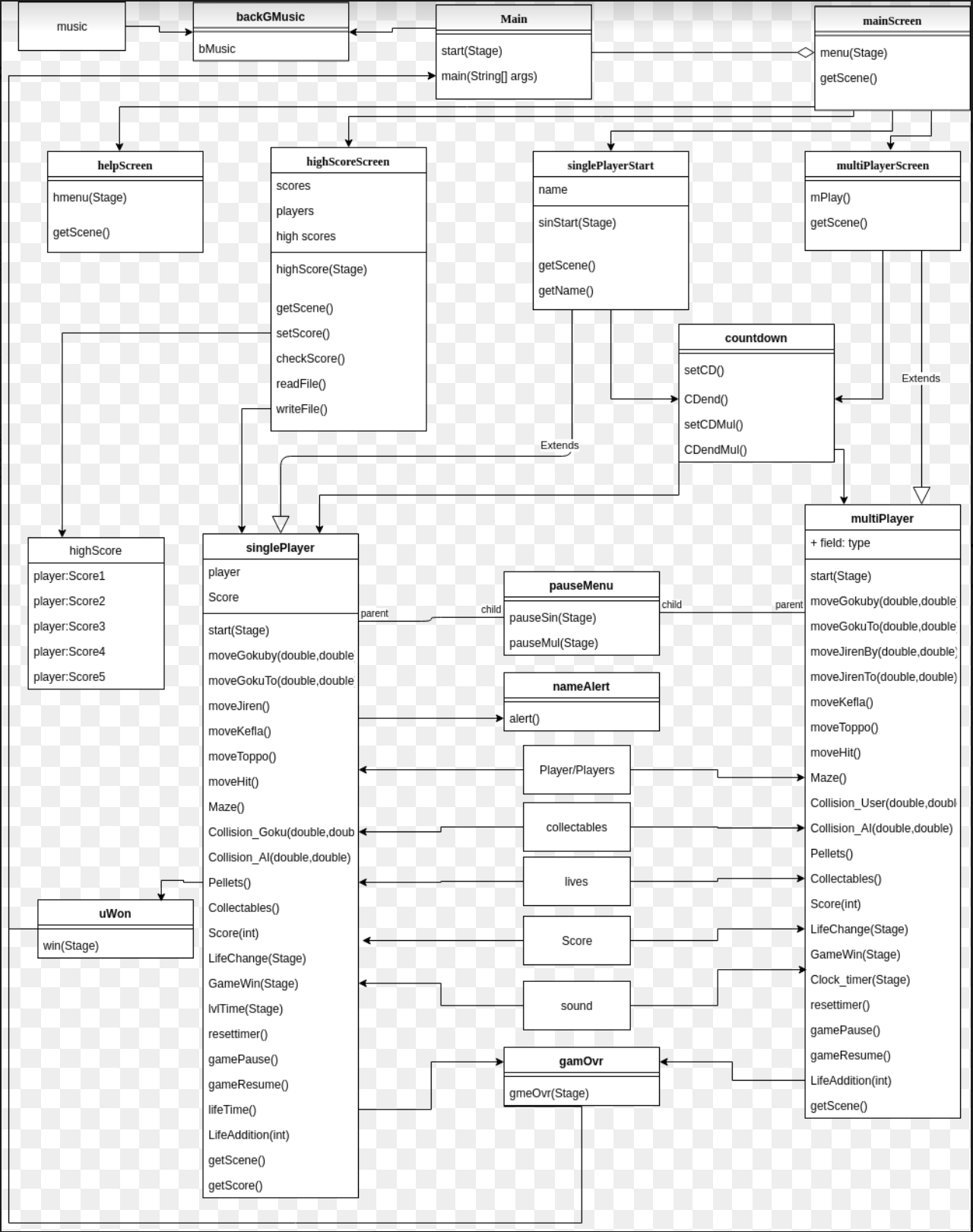


**Flowcharts highlighting AI method:**









Class Diagram