TECHNICAL DOCUMENT

Java Game Development

By Nipoon Patel and Suhan Muppavaram

Table of Contents

Summary

The brief of this project was to develop a game, similar to that of the original “Combat” game made by Atari in 1977. Our client has demanded that the game should be made for his eight year old son. We are given a set of minimum requirements which we are required to fulfill, but at the same time, we need to make the game fun. After the game timeouts (after 2 minutes), a congratulations screen should be shown to the player. The end goal of this project is to make a game that is aesthetically pleasing, exciting for the user, and most importantly, fun to play.

Meeting the Requirements

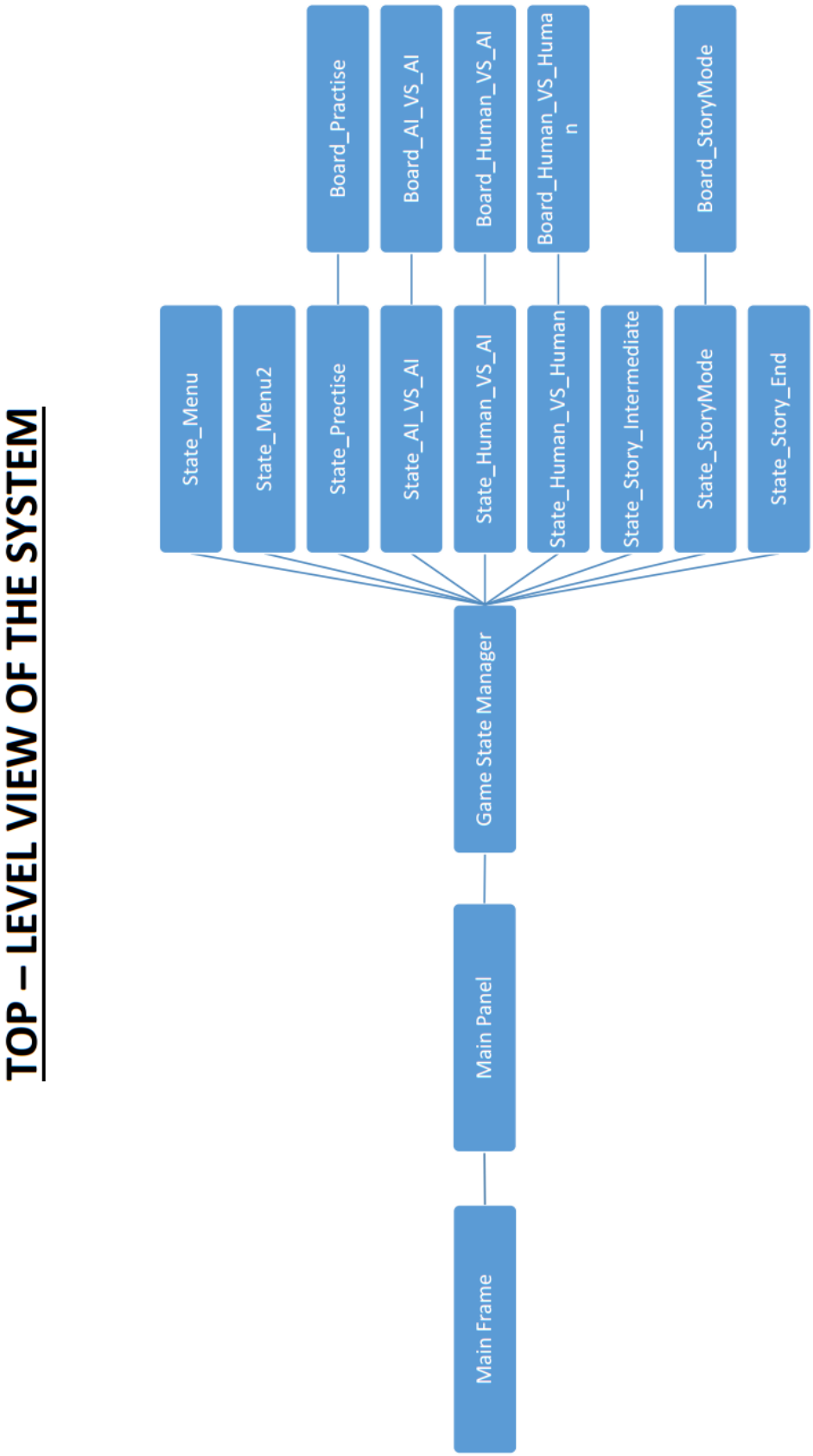
The game we have created is simple to play and navigate, through the menu screens. Whilst making this game, we kept in mind throughout its development that it is for an 8 year old. We decided to make it less flashy and more to the point of what the user requires. When you first run the game, a cool animation is displayed, welcoming the user to Combat 2.0. If any form of the game is started, a countdown timer is displayed while also showing the relative positions of the tanks and obstacles. The point of this is so that it gives time for the player to understand the layout and plan a route of attack.

The local multiplayer mode is straight forward with one player using WSAD controls and the other using arrow keys. Power ups are displayed in the top corners by the players’ names. Game lasts for 2 minutes as desired, and the player with most points is declared the winner at the end of it. Pressing p will pause and un-pause the game, as required.

The single player mode has all the same features, but the AI will try to attack you.

The practice mode is also the same, but the opposition “tank,” or target in our case, is stationary, as required.

Top Level Diagram



Development Issues

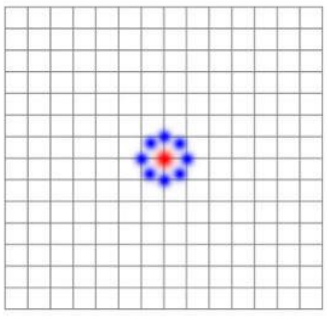
Developing the AI was quite difficult in the initial stages. We had to first look up a decent path finding algorithm to track a path to the target, and make the tank follow that path. In the beginning, we were originally moving the tank every pixel of the 1024*768 pixels on the screen and this took far too long for the AI to move.

- The computation took way too long, with 30 fps we had to have the computation time less than 33 ms, but it was talking up to half a second.

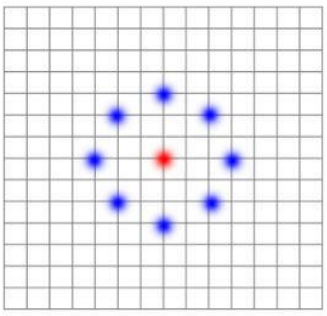
To overcome this, we decided to make the grid layout.

- Each grid was made of multiple pixels
- In normal speed the tank moves at 3 pixels per frame, with speed power up it moves at 5 pixel per second.
- Also we made it so that, the new path of not calculated every iteration, but happens every 10 iterations. This makes the game lag much less.

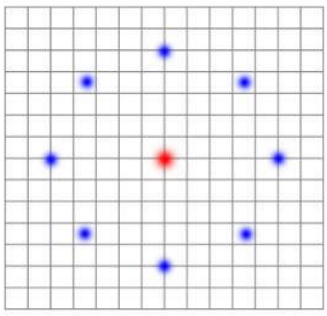
The following grids show the movement of AI tanks.
Red dot represents the initial position.
Blue dots represent the possible final position.



Slow speed powerup
Grid size : 1x1 pixel



Normal speed
Grid size : 3x3 pixel



Fast speed powerup
Grid size : 5x5 pixel

Another issue that was faced during the development, was transitioning between screens. Initially, we were using multiple JPanels that stack on top of each other for each different game state. However this didn't work out, because as the panels stacked, the main panel that was on display was slipping down the JFrame, and after multiple transitions from the panels, the panel on display would disappear. To overcome this, we decided to resort to a Game State Machine method. This utilizes only one JPanel and one state is displayed on it at a time. Transitioning between states is much smoother and JPanel stacking was no longer an issue.

Getting the tanks to rotate properly was also fairly difficult in the beginning stages. The problem was that if we rotated the image, the entire co-ordinate system (or grid) would rotate with it. In the end, the fix was easy. All we had to do was store the original co-ordinate system in an Affine Transform variable and after rotating the tank, set the co-ordinate system back to what we had stored in the variable initially. But finding the fix was very difficult.

Features

- We decided to have a map selection menu so the user can practice on a few set maps to get used to them and better their friends in the game, or we give them the option to play in a random map.
- We let the users input their name.
- We made an animated main menu screen to entice the user.
- We made a proper AI that follows the A* path finding algorithm, picks up power ups that are in range, and shoots when in range of the enemy tank.
- Implemented an AI vs. AI mode.
- A fun story mode.
- Timer changes color and flashes as it gets closer to the end of the game.
- Explosions.
- Added extra power ups.

Sustainability of Tools

The java language overall was simpler to pick up than the other slightly lower level languages like C++, and even C. In terms of learning a popular modern day language, java is important to learn as basics maybe required in the work force. Developing a basic game in java is a good way to get to grips with the language. However in terms of game development, we wouldn't say it's the best and easiest to develop such an application. One of the main complaints we have is that we cannot explicitly delete an instance of an object after it's made. Only when that object is completely dereferenced, the garbage collector removes it from the program which is not easy to do when developing a game. The consequence of this, is that a certain program can be running in the background and consuming CPU usage when not necessary and slow down the program.

Possible Improvements

For the future, we could manage or code better, so that it runs more smoothly. At the moment, there is a significant amount of lag due to the amount of operations it needs to perform in the background. We also could of utilised inheritance. We have a lot of similar code in or board classes, and if we had made a general board class and make the different board classes extend off that, it would make our code a lot cleaner. However, we did have some inheritance with our general class. It would have also been beneficial for us to break up the different aspects of our game codes into different packages and import those packages when necessary. For e.g, the game states could've been a package and the sprites one package and the boards etc. This would help organise our code and ease of access would've been better. Another improvement could be to utilise or git stuff better. We would always run into merge conflicts when pushing our code and we realised fairly late that if we made changes and pushed to git, the other person would receive only those changes and not the entire thing. For our game logic improvements we could have improved on our collision with walls. At the moment, the tank will just stop against a wall, but in the future, we could try to get it to slide across the wall rather than fully stop so we don't lose the momentum of the game. Also destroying walls could be another fun aspect to add for the next time we create a similar game.