Triple Triad: Design Document

Quinten Lambert

October 3, 2016

Contents

1 Vision and Scope

1.1 My Vision

The focus of this project will be the creation of a card game reminiscent of the Final Fantasy VIII mini game Triple Triad in the Unity Game Creation engine. Through this creation of a project in this environment I am attempting to increase my knowledge of programs used to create video games. My vision is to gain the skill set necessary to create a game in the Unity environment so that I can pursue my dream of developing video games. This project provides the first stepping stones that will allow me to approach that vision. The project will be a basic card game played on a three by three grid, that will pit the player against an AI player that will analyze the state of the game board to make the best play possible.

1.2 The Scope of my Project

While I could recreate the entirety of the Triple Triad game, that is not my goal. The projects focus is on algorithm analysis for the the AI system of the game. With this in mind, my project must have a working AI by the end of the semester. For that to work, the project must also be a functioning game that a player can play against the AI. This includes the creation of the following aspects of the game: card objects that the players will be using, a game board for the game to be played on, and a system that will handle the turn progression of the game with a win condition built in. Other features could be included but they are not necessary considering the current scope. This may include features such as: a deck of cards that both the player and the AI choose from to populate their hands, different sets of rules that change the way the game is played, and the creation of a difficulty setting that will change how the AI reacts.

2 Learning Goals

This project provides an ample challenge to me in the form of understanding how to use a new program as well as the understanding of algorithm analysis for the AI system. These two major challenges will make up the focus of the learning goals of this project. In either case, my knowledge is lacking and I am using this project to remedy that. With that in mind, the following questions will define what I expect to be learning from this project.

1. How can I understand the capabilities of the Unity engine through example? (Understanding)
2. How can I effectively analyze AI algorithms to create the best possible outcome? (Implementation)
3. How do I compile my results and display them effectively to prove my theories? (Reflection)

2.1 The Values of Understanding

The understanding aspect focuses on the understanding of the environment that I will be working with. The Unity engine is mostly new to me, and even though it uses C# as a base coding language, it is heavily modified to suit the program. Due to this I expect to run into issues understanding what the Unity engine is capable of, which could set me back depending on the situation.

2.2 The Focus of the Implementation

The major focus of my project will be the analysis of AI algorithms to determine the best one to use in my game. This testing and implementation of these algorithms encompasses my second learning goal. At the moment I am mostly unaware as to how I am going to test these different algorithms to determine their usefulness in my project. I find this particularly important since I will have to do a good deal of research to both understand how to create,

implement, and analyze each algorithm that is a contender to be used in my program.

2.3 Reflections: The Importance of Clarity

Finally, the reflection portion of this project will be highly experimental on my behalf. I have not publicly released my processes on any previous project, and to form blog posts around this project will both be new to me as well as feel awkward. Understanding the language, tone, and word choices to make in these posts will constantly be a large problem of this process. Another issue I foresee happening is completing drafts for publication if I run into problems writing them. My goal with this section is to overcome these issues and provide accurate documentation of my project through the usage of blog posts and reports throughout this semester.

3 Project Requirements

☐ **MUST have at least 5 unique card objects/sprites**

- ☐ Each card MUST have the 4 integer values ranging from 1-9 that represent that cards power for each of the 4 cardinal directions
- ☐ Each card MUST have a string value that represents which player owns that card (a card MUST be owned by a player)
- ☐ Each card MUST be visually colored to show who owns the card on the game board (ex. a red card belongs to player 1, a blue card belongs to player 2)
- ☐ Each card SHOULD have an image that will differentiate it from the others

☐ **MUST have a three by three game board**

- ☐ Game board MUST have 9 possible locations to play cards
- ☐ SHOULD be a sprite covered in a graphic

☐ **Project MUST be a working game**

- ☐ MUST have the ability to play cards from hand to the game board
- ☐ MUST Handle Turn progression
- ☐ MUST Compare the values of Cards
- ☐ MUST Have a win condition
- ☐ MUST be able to handle 2 player local multiplayer
- ☐ SHOULD include a title screen to determine whether to play local multiplayer or against the AI
- ☐ SHOULD have animations to depict movement of cards

☐ **There MUST be a working AI system**

- ☐ MUST make use of a complex algorithm that implements the game tree to analyze all possible game states (ex. min-max, negamax, A*)
- ☐ MUST analyze current board state
- ☐ MUST make the best play based on cards in hand, cards in the player's

hand, and the board state

- ☐ The AI SHOULD be beatable

☐ **There MUST be a User Interface**

- ☐ There MUST be a visualization of how many cards each player owns through two numbers on the interface that should both add up to 10
- ☐ There MUST be a pointer object to allow the player to make selections
- ☐ There SHOULD be a visualization of the cards in both players hands on the screen

☐ **SHOULD be visually appealing (not just blank blocks floating in a void)**

- ☐ SHOULD add card textures to depict the different cards
- ☐ SHOULD include a board graphic (picture to represent the board)

☐ **Project SHOULD compile and run on multiple different systems (MUST run on at least one of the following)**

- ☐ MUST run on Windows (testing operating system)
- ☐ SHOULD run on Linux
- ☐ SHOULD run on Macintosh
- ☐ SHOULD run on IOS
- ☐ SHOULD run on Android

4 Architectural Overview

This project must first start off with a base to work with. The base for triple triad revolves around the cards that the players play with, and the board that they play on. These two elements of the game are absolutely necessary in order for the implementation can progress. Beyond that, the flow of the game must be implemented to allow for the game to be fully functional. The final step will be to add the AI to the already working game.

4.1 The Card Object

In the Unity engine, the card objects will be represented by a list of sprites, which will serve as a visual representation of these cards within the game space. Each of these cards will have a unique identifier, four integer values that represent the power of that card for each cardinal direction, an ownership value to show which player owns the card, and a link to card sprite. These cards will be stored in a list format with the elements being organized as so:
[Unique Identifier, North, South, East, West, Ownership, Sprite]
This list will be stored within another list within the program so that these cards can be divided out to the players. To clarify, these cards will be pre-generated and static, they are not subject to changes.

4.2 The Game Board

The game board for the Triple Triad game is similar to that of tic-tac-to: it is a three by three grid that has 9 playable locations. These nine locations will be held in a list, so that they can be called on quickly. Since this list will only have 9 elements, performance will not be affected no matter how the program searches through it to locate data.

4.3 The Game Flow

Every game has a specific way that it moves from each state of game play. In this instance there are a total of 8 states of play. Figure 1, below, shows these eight states.

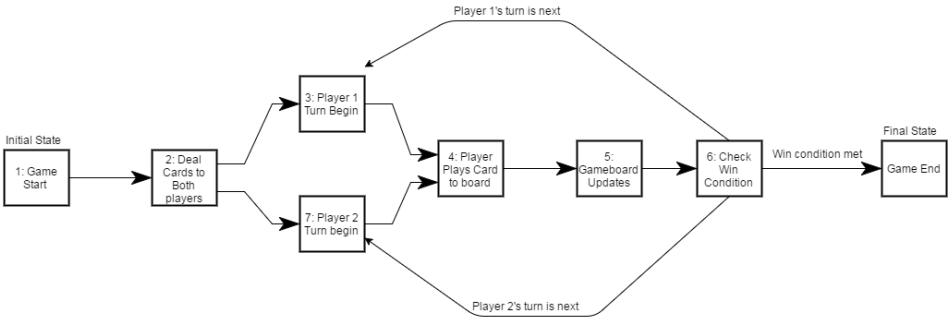

Figure 1: The state diagram for the flow of the game

State 1: The Start of the Game

This is the initial state of the game. It will consist of the opening menu that will allow the player to have two choices: to play against a friend (local multiplayer) or against the AI. Once the player has made a decision, they will move on to state 2.

State 2: Dealing the Cards

Once the game has started up, both players will receive five cards that will be placed onto the game board. All ten cards will be visible to each of the players, so that they can see what their opponent is capable of, and strategize appropriately. When both players have their 5 cards, the game moves onto state 3 if player one goes first, or state 7 if player two goes first.

State 3: Player 1 Turn

This state is only encompassing the players analysis of the current state of the board, and their cards in hand in anticipation as to what they will play. If this player is the AI, this is when the algorithm will run and the game state will

be processed. All possible moves will be recorded, and placed into a tree data structure. Once the move has been thought of, proceed to state 4.

State 4: Play a card to the Board

This is where either player will place their card of choice from their hand to the game board at a location of their choice. The card will be removed from the list of cards that is the player hand, and will be added to the game board list. Continue to state 5.

State 5: Update the State of the Board

Once the card is played to the game board and the board list has been updated, the next step is for the game to compare the values on the board and see if the player that has played the card gets any new cards through the beating cardinal values. When all cards have been switched to their new, or previous, owner, proceed to state 6.

State 6: Check the win condition

The win condition of this game cannot be met unless the game board has been filled, if the game board has not been filled, the game will proceed as normal with the next players turn and shall return to either state 3, or state 7. If the board is full, the game will analyze how many cards each player has and will proceed into state 8.

State 7: Player 2 Turn

See State 3

State 8: Game Over

The final state of the game will take the input from state 6, and will conclude the game displaying the winner on screen, or if no one won, show that the game was a draw. End Game.

4.4 AI Implementation

As the previous section stated briefly, the AI algorithm will take the place of one of the players. This AI will be armed with the knowledge of all ten cards that are on the game board, and the state of the game board at all times. In order to achieve this goal, the AI will be running off a version of a Game Tree algorithm. A Game Tree will hold each of the possible moves that the AI can take as the leaves of the tree data structure, while the nodes will represent different states of the game. This process is shown below in Figure 2 by breaking down a game of tic-tac-toe.

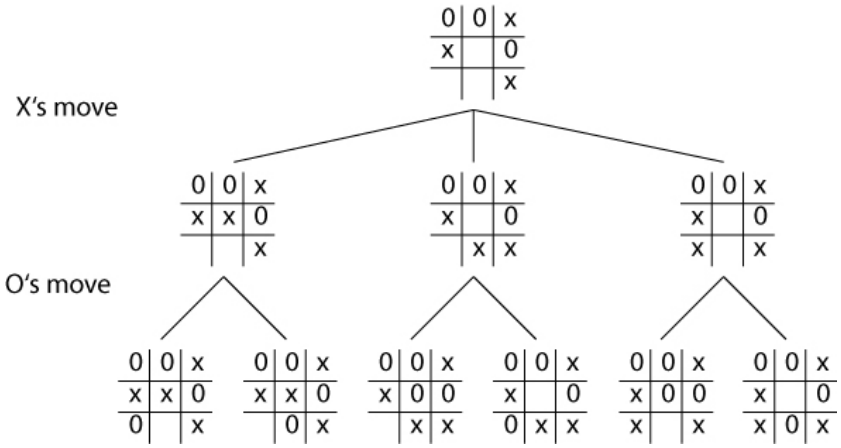

Figure 2: A visual representation of a Game Tree when used on Tic-Tac-Toe

Using a similar method as shown above, the algorithm that the AI will run in this game will have to be capable of checking which cards are cardinally adjacent to the current play. When checking each possible move that the AI could make, a game tree will be populated with an integer value that will represent how good that play would be. At this moment I am unsure as to exactly how this algorithm will work, however, the usage of a game tree is a must. I am unsure at this point due to the fact that I have not chosen a specific algorithm to implement, I still need to put research into which one will work best for this

project. This is due to its versatility to hold both the multitude of possible game states, as well as every possible move to transition between each game state. Eventually the AI will read a leaf node on the outside of the tree, and with that they should win the game.

5 Development Plan

At this moment I will only have a little over half of the semester to implement this project. With this in mind I propose the following roadmap:

5.1 Within Scope

The following items are mostly repeated from earlier, but now will have timestamps added to them as well as saying how long in weeks I expect to work on them. This is assuming that there is 8 weeks of the semester left to work with, and working approximately 10 hours on the project a week.

Week 1 - 1.5

☐ **MUST have at least 5 unique card objects/sprites - total 15 hours**

- ☐ Each card MUST have the 4 integer values ranging from 1-9 that represent that cards power for each of the 4 cardinal directions - **3 hours**
- ☐ Each card MUST have a string value that represents which player owns that card (a card MUST be owned by a player) - **2 hours**
- ☐ Each card MUST be visually colored to show who owns the card on the game board (ex. a red card belongs to player 1, a blue card belongs to player 2) - **7 hours**
- ☐ Each card MUST be stored in a list to be called upon later - **3 hours**

Week 1.5 - 2

☐ **MUST have a three by three game board - total 5 hours**

- ☐ Game board MUST have 9 possible locations to play cards held in a list data structure - **5 hours**

Week 2 - 4

☐ **Project MUST be a working game - total 20 hours**

- ☐ MUST have the ability to play cards from hand to the game board - **5 hours**
- ☐ MUST Handle Turn progression - **2 hours**
- ☐ MUST Compare the values of Cards - **5 hours**
- ☐ MUST Have a win condition - **2 hours**
- ☐ MUST be able to handle 2 player local multiplayer -**3 hours**
- ☐ SHOULD include a title screen to determine whether to play local multiplayer or against the AI -**3 hours**

Week 4 - 6.5

☐ **There MUST be a working AI system - total 25 hours**

- ☐ MUST make use of a complex algorithm that implements the game tree to analyze all possible game states (ex. min-max, negamax, A*) - **5 hours** (to make a final determination on which algorithm to use)
- ☐ MUST analyze current board state and make the best play possible (The actual implementation of the algorithm) - **20 hours**

Week 6.5 - 7.5

☐ **There MUST be a User Interface - total 10 hours**

- ☐ There MUST be a visualization of how many cards each player owns through two numbers on the interface that should both add up to 10 - **6 hours**
- ☐ There MUST be a pointer object to allow the player to make selections - **4 hours**

Week 7.5 - 8

☐ **Project MUST compile and run on Windows - 5 hours**