**Final Report Documents-Zombie Killer**

For CS6150/Game AI/Spring 2017

Table of Content

[**Basic Information**](#_26n92e7ig1zb) **2**

[**Title of Project**](#_7yndu2vozq3h) **2**

[**The title of this project is called Zombie Killer.**](#_27uonhtew8ia) **2**

[**Project Team Plan**](#_dh47ep4k3lbz) **2**

[Task Allocation](#_t6zrmdth748x) 2

[Detailed Project Plan](#_nud00j456gv5) 3

[**Overview of Project**](#_fuenqrprhx0c) **3**

[**Rule Detail**](#_ehqb8t397090) **4**

[**Vision Statement**](#_gwiwiihtg95h) **6**

[**System Features**](#_x798otydlr8x) **8**

[Game Engine](#_gfpcjkhko9ke) 8

[Decision Tree](#_8pb45w6cn5w5) 13

[Monte Carlo](#_y9brogve1zr9) 15

[Max-Min](#_b8qoya4dl7m2) 17

[**Tools and Libraries**](#_12rlnsy0795) **17**

[**Video**](#_3vaorxz3jwzp) **18**

[**Codes**](#_g2fj2jz7cvmb) **18**

[**Result Comparison**](#_vm16zymxfq6z) **19**

[**Reflection**](#_4co0rynsjm3l) **21**

# Basic Information

Our team is called ZK-Team.

# Title of Project

# The title of this project is called Zombie Killer.

# Project Team Plan

## Task Allocation

* **Jialin Gao:** project manager
* **Yiren Ding:** development engineer
* **Tianhao Fang:** development engineer
* **Shubhi Mittal:** development engineer

Detailed task allocation appear on project plan.

## Detailed Project Plan

<http://publish.smartsheet.com/676ba7f339334b6884a4fb49573453a7>

The detailed plan is changed from our previous proposal. When we started our project, we found that we could not run tests repeatedly on that online compiler. In addition, it does not support compiling codes in separate files. That is difficult for us to debug our codes. As for the Q-learning, we think it is not suitable for this game， because the key to get high score in this game is predict the future state. Therefore, we discard Q-learning stage and add new MaxMin stage as substitution.

# Overview of Project

The game we created is a killing-zombie game. There is only one agent who has a powerful weapon in this game. He needs to protect humans from being killed and kill all zombies around in a closed map.So in our project, we will be using game AI techniques learned in class and other techniques via doing researches to make the only agent accomplish his goal. In addition, we will try to figure out which kind of movement of the agent can maximize the score the agent can earn (the agent can earn higher scores by killing multiple zombies in succession, and the lost of human will reduce score dramatically). The AI strategy we used in this project includes decision tree, maximin algorithm and Monte Carlo algorithm. These will be explained later in the section of vision statement.

# Rule Detail

The game is played in a zone 16000 units wide by 9000 units high. You control a man named Ash, wielding a gun that lets him kill any zombie within a certain range around him.

Ash works as follows:

1. Ash can be told to move to any point within the game zone by outputting a coordinate X Y. The top-left point is 0 0.
2. Each turn, Ash will move exactly 1000 units towards the target coordinate, or onto the target coordinates if he is less than 1000 units away.
3. If at the end of a turn, a zombie is within 2000 units of Ash, he will shoot that zombie and destroy it. More details on combat further down.
4. Ash could only see the surroundings (zombies, other people) within 3000 units

Other humans will be present in the game zone, but will not move. If zombies kill all of them, you lose the game and score 0 points for the current test case.

Zombies are placed around the game zone at the start of the game, they must be destroyed to earn points.

Zombies work as follows:

1. Each turn, every zombie will target the closest human, including Ash, and step 400 units towards them. If the zombie is less than 400 units away, the human is killed and the zombie moves onto their coordinate.
2. Two zombies may occupy the same coordinate.

The order in which actions happens in between two rounds is:

* Zombies move towards their targets.
* Ash moves towards his target.
* Any zombie within a 2000 unit range around Ash is destroyed.
* Zombies eat any human they share coordinates with.

Killing zombies earns points. The number of points you get per zombie is subject to a few factors.

Scoring works as follows:

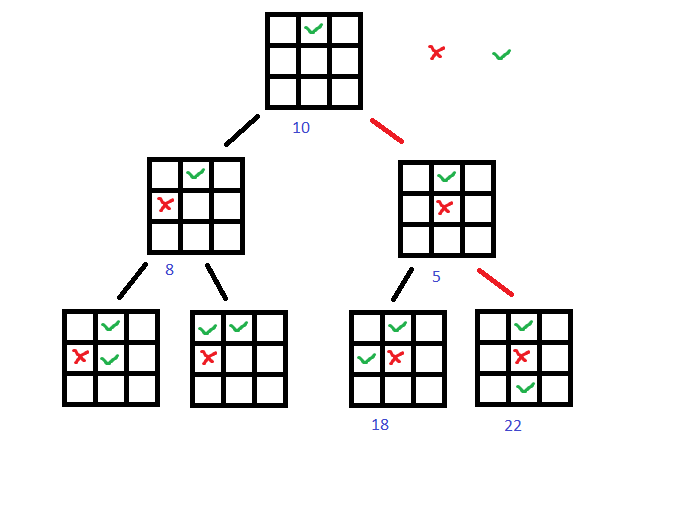
* A zombie is worth the number of humans still alive squared x10, not including Ash.
* If several zombies are destroyed during on the same round, the nth zombie killed worth is multiplied by the (n+2) th number of the [Fibonacci sequence](https://wikipedia.org/wiki/Fibonacci_number) (1, 2, 3, 5, 8, and so on). As a consequence, you should kill the maximum amount of zombies during a same turn.

# Vision Statement

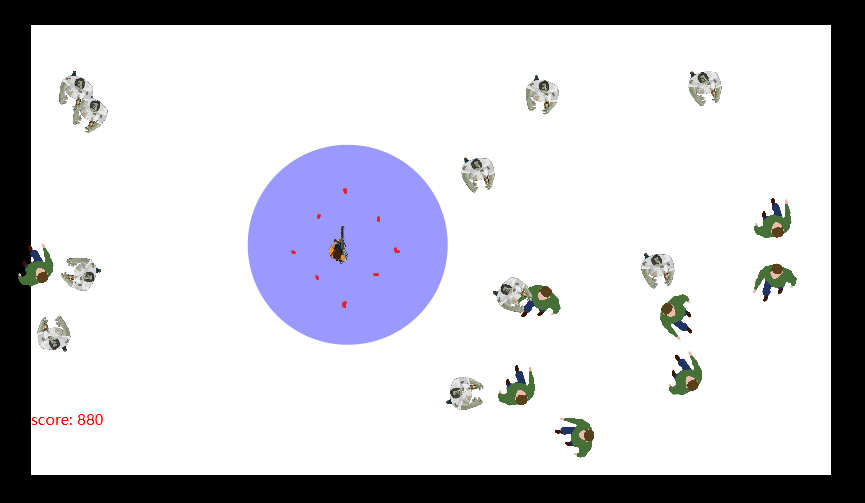
Our project can be divided into three stages. Each stages is designed for one AI strategy. In first stage, our primary goal will be go through all test cases, regardless how many score we can get. Decision Tree is commonly used to identify a better strategy to reach the goal. We plan to use predefined decision tree in our first step, since it could be easy and fast to test all of our ideas. We will design strategy and movement for possible situation. So we assume in stage 1 we can reach victory result after running all test cases. Compare to assignment 3, in this game, the agent have more complex task and will encounter more tricky situation than agent in pacman. We did a research on test cases they provide, they have comprehensive strategies to test our algorithm. For example, they have different test cases for different number of overall zombies, zombies comes from all different direction, human that we need to protect stay together or separated in several clusters. So the pattern of zombies are much more complex and has more combination than ghost in Pacman.

In second stage, we will focus on Monte Carlo algorithm. Monte Carlo algorithm is an algorithm used to simulate the behavior of systems. It makes use of randomness and statistics to get result. It means it is a random strategy for this project. The computation process uses random numbers to produce an outcomes. Probability distributions are assigned to inputs. Therefore, it is not an exact method. Sometimes the game that uses Monte Carlo algorithm may not pass all tests. Monte Carlo algorithm relies on repeated random sampling to obtain numerical results. This method cannot guarantee finding optimal solution. The only thing we can ensure is that with the increasement of sample size, the result will be close to optimal solution. It is also a smarter strategy than the stupid decision tree. In addition, it can get higher scores if it can pass tests. It can get some of zombies together and kill them. Since if the agent are able to kill lots of zombies in one turn, it will gain a lot of score at once(see game rules). In our project, if we randomly put dots in game setting and we consider each dot as our agent’s target, the agent will select the move that will get highest score next scene. To avoid the problem of large setting, we decide to place dots in a relatively small range. Therefore, not multiple random dots will give us same score.

The last stage focuses on the MaxMin algorithm. MaxMin algorithm is widely used in cheese game and other turn-based game. The algorithm is used to make prediction about future actions of both player and opponent. The algorithm start from current state that players take all possible actions and then opponent take all possible actions based on each action that is taken by player and loop until reach the depth limit. After each agent take an action, the new game status is evaluated by an heuristic function. The heuristic function takes the game state and returns a number indicating the advantage of player as well as the disadvantage of opponent. The possible actions and the related game status form a prediction tree. In this tree, only the branch that player make the heuristic function become maximum in his turn and opponent make the heuristic function become minimum among with all the actions in the same turn will selected. By doing such search the player is do the best possible action he can ever take within the depth of search tree.



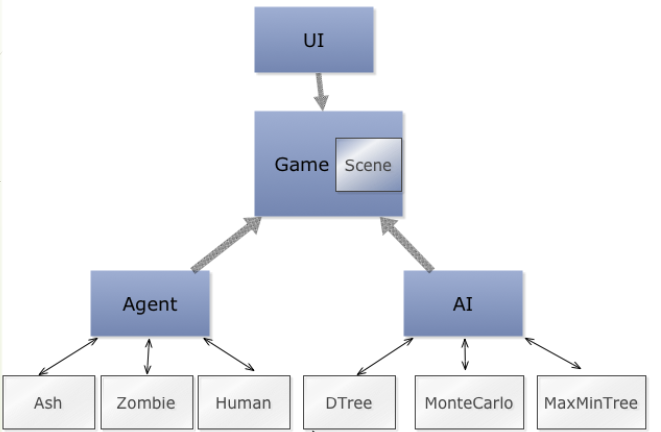
The maxmin algorithm can apply to zombie game since the game is also a turn-based game. However different to cheese and other games, the action of ash in zombie game is choose a point in a circle that has center in ash’s position and radius of speed of ash. The point in a circle is almost countless, so we could not find all possible moves like play other game, so we need to simplify the possible actions. In the zombie game we limit the possible moves of ash only to 8 points on the boundary of the range that ash can move. Also we apply a heuristic function that evaluate both the distance ash to nearest zombie, the amount of human remains on scene, and the score get. We use the maxmin algorithm to find the best solution that has the highest heuristic value.



# System Features

## Game Engine

The main architecture of our project is shown as figure 1. There are four main components in this architecture. Since we decide not to use online compiler, we have to create our own user interface for our project. The second component is Agent. There are three kinds of agents, Ash, zombies and human, in our project. Each one is defined in separate classes. AI component includes three main AI strategies: decision tree, monte carlo, and maxmin. The last component is Game. It illustrate the basic rules of our game. The scene inside Game explains how agents move and how scenes change in each turn. Our tests are also inside Game component.

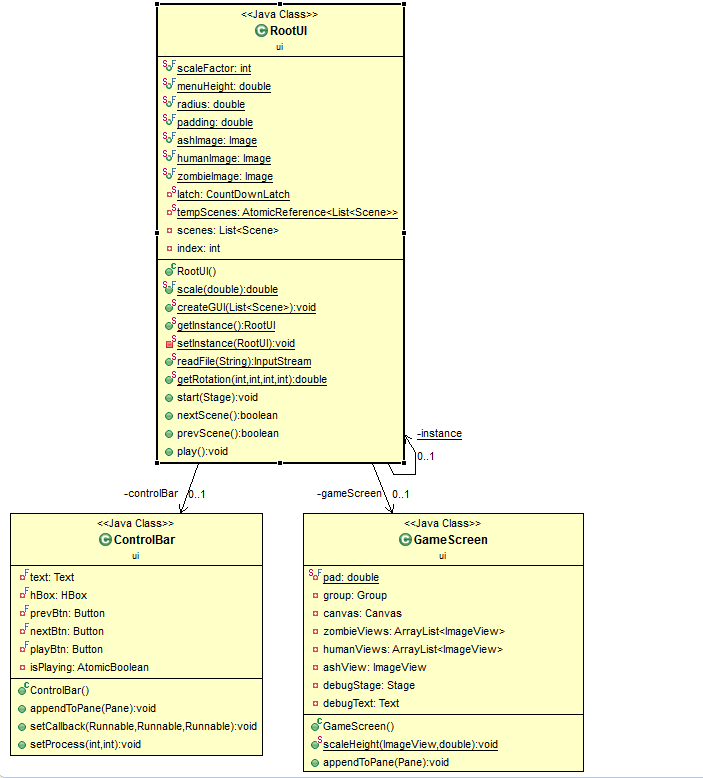
****

**Figure 1 Architecture of System**

The user Interface is shown in Figure 2. The agents in green color are human. They are static and cannot move to anywhere. In order to win the game, at least one human is still alive when all zombies are killed. The agents in white color are zombies. They can move to closest human or our main agent Ash. The agent that holds a gun is our main agent Ash. His goal is to kill all zombies and protect human at the same time. The purple circle around Ash is his shooting range. Once zombies are located inside this circle, they are killed immediately. The control bar is at the bottom of this screen. Users can use Prev and Next buttons to play each scene. Or, they can choose to use Play button to automatically run the game. The Figure 3 shows the class diagram of UI implementation.

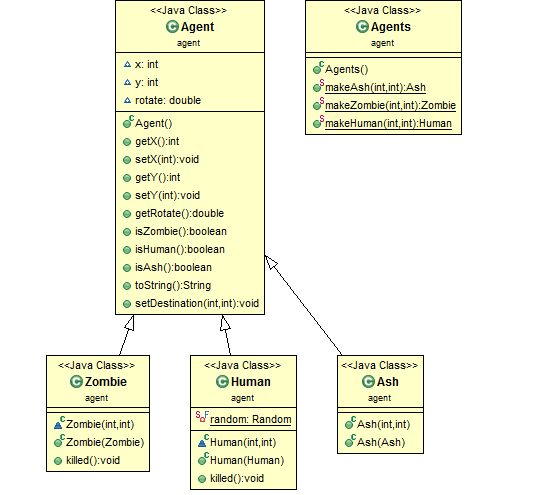
****

**Figure 2 User Interface**

****

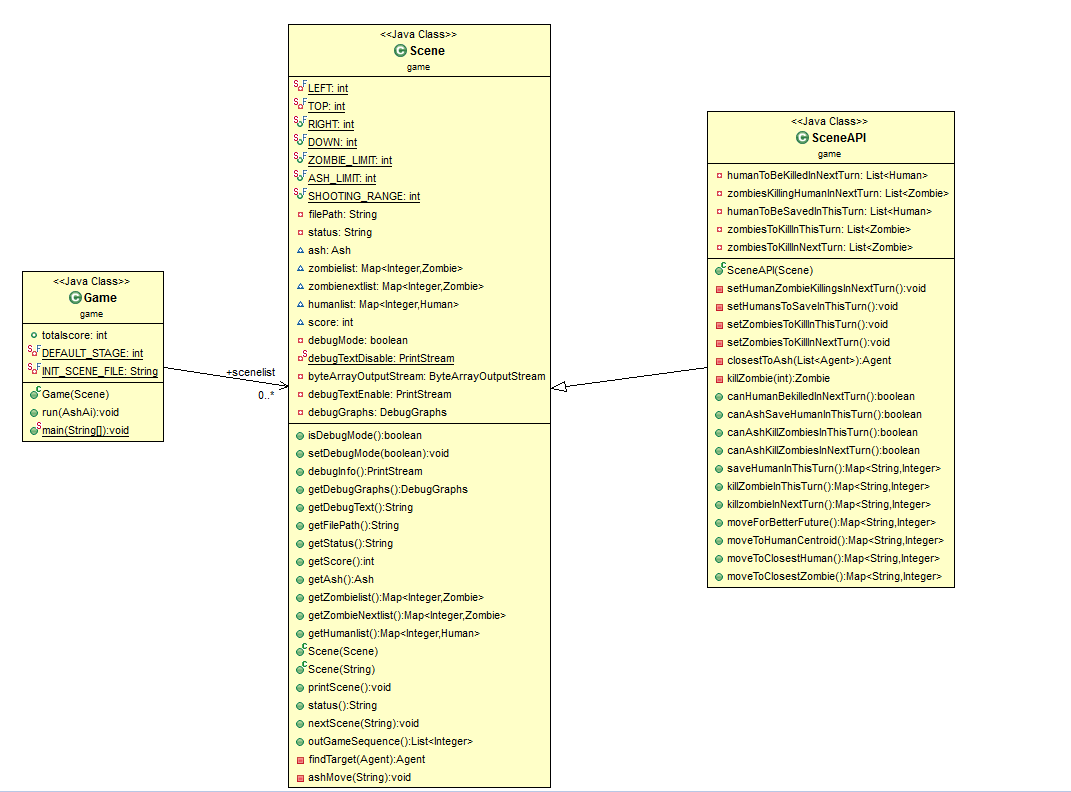
**Figure 3 Class Diagram of UI**

The Figure 4 shows the class diagram of Agent package. Three exact agents, Ash, human and zombies extend from Agent class. Each one has their own constructor. There are three static methods in Agents class creating new objects for those three agents, which uses the static factory design strategy here.

****

**Figure 4 Class Diagram of Agent**

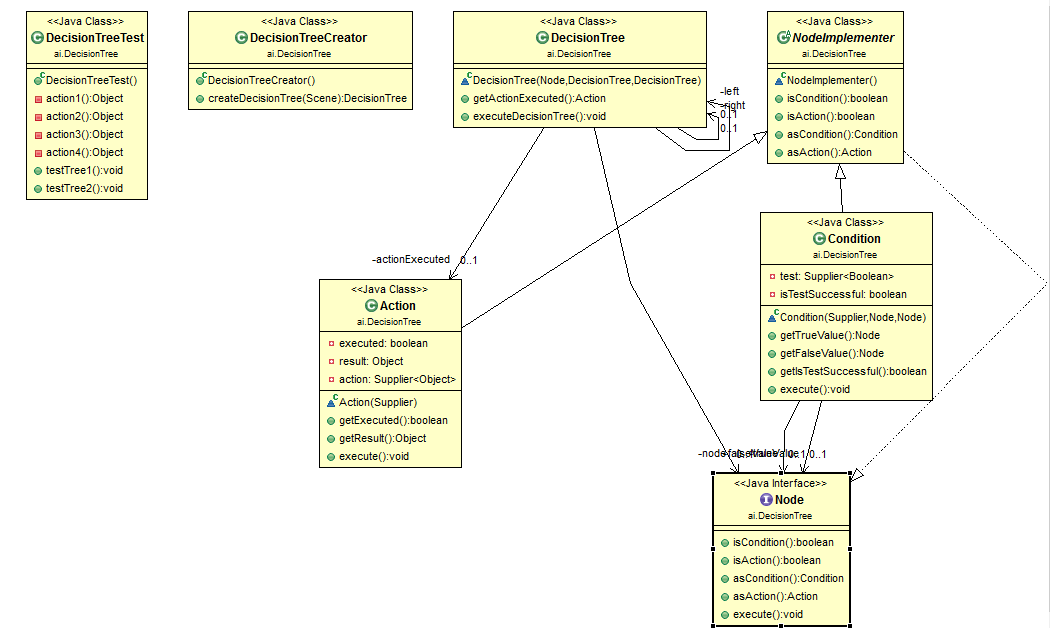
The Figure 5 shows the class diagram of Game package. Scene is the main class in this package. In the class of Scene, several methods are defined. They include how zombies move to human or Ash, how Ash or zombies move in next scene, and how game rules are implemented in this game. All JUnit test cases are defined in separate files, but they can be accessed in Game class.

****

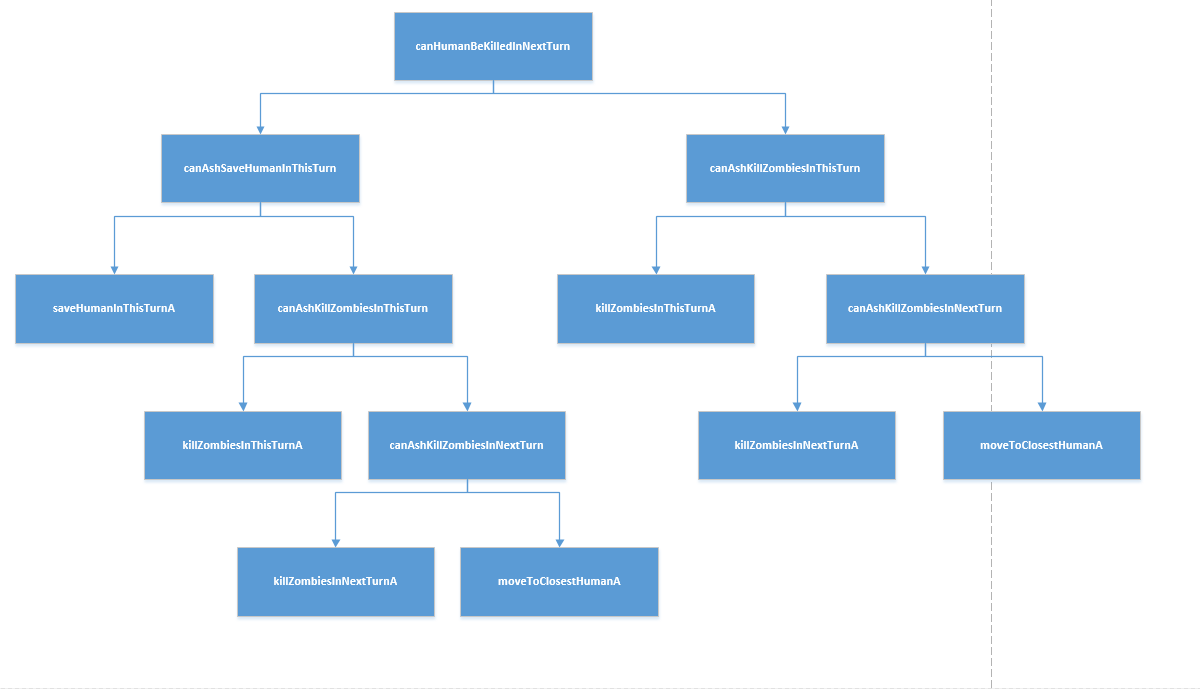
**Figure 5 Class Diagram of Game**

## Decision Tree

The Figure 6 shows the class diagram of Decision Tree package. In this package, Node is an Interface for all decision tree nodes. Action and Condition are two kinds of Decision Tree nodes which implement the abstract method in the Interface Node. DecisionTree class defines the constructor for DecisionTree. An entire decision tree is built within the class of DecisionTreeCreator. Figure 7 is the diagram of built decision tree. That tree starts from node canHumanBeingKilledInNextTurn. All nodes which name ends with ‘A’ are action nodes. Others are condition nodes.

****

**Figure 6 Class Diagram of Decision Tree**

****

**Figure 7 Decision Tree Structure**

## Monte Carlo

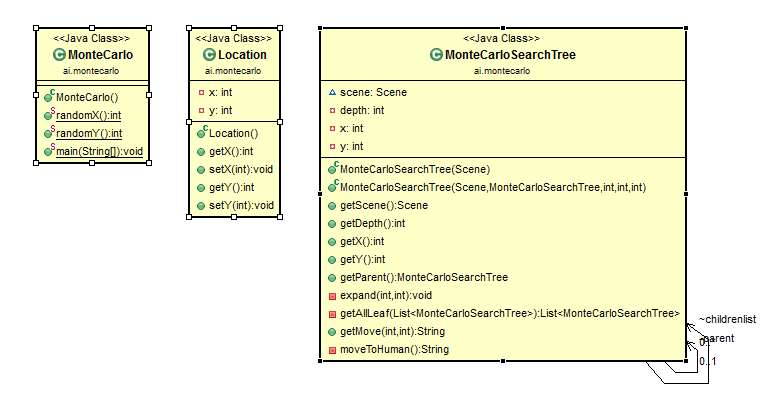
The figure 8 show the class structure of Monte Carlo search tree. The Location class is a class simply record X, Y coordinates for convenience.

In MonteCarlo class, it defines several static function to generate numbers for random x and y. We tried several strategies here. The first one is generate x in 0-16000 and y in 0-9000 which are the scene boundaries. Second, we tried calculate x and y within a certain range that centered by Ash. We tried circle(radius is the 1.4 or 2 times of moving limit) and square(side length is 1.4 or 2 times of moving limit ). It doesn't affect the result dramatically if the sample size big enough. We eventually choose square with 2 times of moving range, which give us a slightly average score.

The MonteCarloSearchTree class define the functions of expand search tree, refine result and return move. We use limited depth, breadth-first search method to explore solutions. Generate the whole search tree with the given depth. Actually we can let the search tree reach the end of game to find optimal move, but since the size of possible move is even much larger than Go game, it is impossible to simulate the game to the end.

When the tree reaches depth limit or the game status reach pass or fail, it will stop expanding. Then it collect all leaf node, and select the node with max value. It is a greedy approach basically. If several leaf have same score, pick the one with smaller depth.

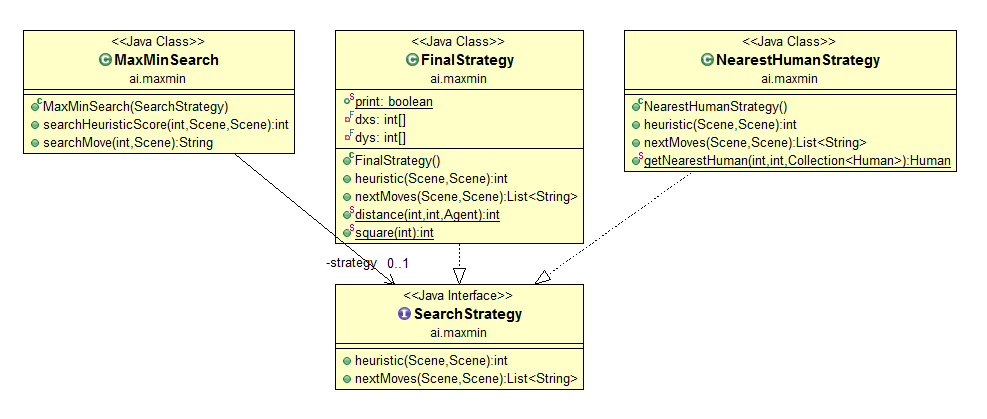
After implementing all above functions, we ideally put 10 500 as parameter for Monte Carlo AI. As a result, we ran out of the memory of laptop. The reason is both time complexity and solution space is O(M^N), which is too high for our input parameter. At last, we set the search depth to 4 and sample size to 100. It is a compromise between performance and accuracy. Also, we decrease the sample size by 2 times for each further depth of search. In this way, it will decrease the probability to find the optimal solution. But it increase the efficiency of this algorithm a lot.

****

**Figure 8 Class Diagram of Monte Carlo**

## Max-Min

Maxmin algorithm is executed by the MaxMinSearch class. The MaxMinSearch instance consume a SearchStrategy interface in the constructor and store it in a field called strategy. The SearchStrategy play a core rule in the maxmin algorithm. The heuristic(Scene, Scene):int method is used to evaluate the heuristic value of Scene in the maxmin, the nextMoves(Scene, Scene): List<String> method is used to get all possible moves for ash in ash’s turn. There are two implementations. NearestHumanStrategy use a naive strategy that only move to closest human. FinalStrategy will evaluate 8 possible moves around ash. The searchMove(depth:int, initialScene:Scene) method in MaxMinSearch is used to execute the maxmin algorithm by evaluating heuristic provided by strategy instance and get all possible moves of ash by nextMoves() method of strategy instance, the maximum depth for search is given by parameter depth and initial scene is provided by parameter initial scene. The result is the move that could get highest heuristic value in the search. searchHeuristicScore() helper method helps to get the maximum heuristic score in a maxmin search by providing a scene and depth.

****

**Figure 9 Class Diagram of MaxMin**

# Tools and Libraries

What we used in this project includes JRE, JUnit, Eclipse, Intellij, Github and Google Drive. Because our project is implemented in Java, JRE is the prerequisite. JUnit is an ideal tool for our testing. Eclipse and Intellij are used according to personal preference, although using the same IDE is preferred in the industry. Github is used for team collaboration. All team members are familiar with this tool and agree to work together on github. Google Drive is used for our documents. All team members can access and modify documents and presentation.

The original game is an online code competition, but the online IDE is too difficult to debug code. We developed the whole game using java from scratch. So we did not use any other libraries or frameworks.

# Video

The following URL is for our demo video.

<https://www.youtube.com/watch?v=VZf2zwl9ai8&feature=youtu.be>

# Codes

The following URL is for our codes on github. All test cases can also be found on github under test case folder.

<https://github.com/zombieProject/zombieGame>

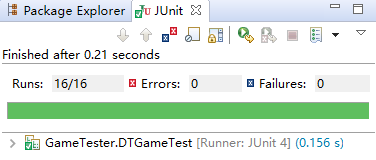
# Result Comparison

## Local Result

We totally have 16 test cases. The result is shown below.

**Decision Tree:**

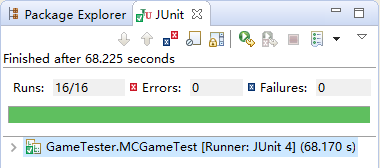
* All test cases pass
* Total Score: 6340
* Time Consuming: 0.156s

****

**Figure 11 Screenshot of JUnit Test of Decision Tree Strategy**

**Monte Carlo:**

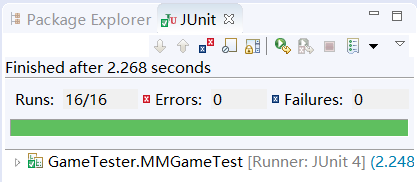
* All test cases pass (not guarantee)
* Total Score: 13980 - 16660
* Time Consuming: 68.170s

****

**Figure 12 Screenshot of JUnit Test of Monte Carlo Strategy**

**MaxMin:**

* Depth = 3
  + All test cases pass
  + Total: 17100
  + Time consuming: 2.587s
* Depth = 5
  + One test case fails
  + Total: 25310
  + Time consuming: 123.282s

****

**Figure 13 Screenshot of JUnit Test of MaxMin Strategy**

## Online Result

We cannot compare our score to the score on leader board, because the real test cases are hidden. The test case they give us is just for practice. To transfer our code to online platform is very difficult, because we have too many classes and functions. We are still working on transfer our code to online IDE to get test score.

For now, we transferred simplest strategy to online platform and run real competition test. The simplest decision tree can get 31740. The top ten score in leaderboard in North America Area is around 80000, which is less than three times better than simplest decision tree. So we assume our approach is very effective since it could have 4-5 times higher score than simplest decision tree.

# Reflection

The workload of this project is much larger than we expected. The main reason is the drawbacks of online IDE make us build a game replica. Developing a game with GUI from scratch requires a lot of work and test.

This project is implemented in Java language. By doing this project, we largely improve our skills of Java programming, especially for game coding and fixing bugs. We learned multiple strategies for decision making discussed in class, such as decision tree, finite state machine, and reactive planning. We implement one of those strategies, decision tree. The concept of decision tree is consolidated.

In addition to decision tree, we learn two new strategies, Monte Carlo and MaxMin. We did a lot of researches on these two topics and finally applied on our project. From those research papers and our experience of implementation. We are not only understand the concept of those algorithms, but also know the advantage and limitation of them. On the process of applying those algorithms in this particular game, we gain lots of experience of how to select suitable approach and how to tune those approach for specific problem.

Although these strategies can be applied, we still need to concern about the performance issue. Using Monte Carlo algorithm can get higher scores, but it spends us much time to run. For some simple scenarios, stupid decision tree may works better than advanced Monte Carlo. In the future game coding, the performance of our produce should be considered significantly. It may affect user experience.

At last, for the future work, we think we can create a hybrid strategy with multiple strategies. This strategy can deal with both simple scenarios and difficult scenarios with qualified performance. We took a research on how Alpha Go deal with performance and efficiency issues, they only use Monte Carlo search tree in a small area where black and white side are in competition. Otherwise, it will use fast rollout strategy to get next move. We think if we can combine a faster algorithm such as decision tree, with MonteCarlo or MaxMin algorithm in the way AlphaGo did, the performance can be improved dramatically.