AI PROJECT FINAL REPORT
COMP 472
Artificial Intelligence
Concordia University

Tawfiq Jawhar
ID 26535186

The Game

Game Rules

1. 2 Player game.
2. Game board 8x8.
3. The larva has to reach the last row on the board for it to win, and the birds needs to prevent that from happening.
4. AI move must take at most 3 seconds.

Play Mode

1. Manual entry for both/one player(s)
2. Automatic entry for both/one player(s)

How to Run the Game?

After running the executable file (exe) of the game, a main menu screen will display 4 images of players:

- Living Larva (human entry of Larva's moves)
- Robot Larva (AI entry of Larva's moves)
- Normal Angry Birds (human entry of Birds' moves)
- Transformers Angry Birds (AI entry of Birds' moves)

Before starting the game, the user has to choose an option for each player (larva and birds). Screenshots of the game are provided at the end of this report.

- Human Larva vs Human Birds
- AI Larva vs Human Birds
- Human Larva vs AI Birds
- AI Larva vs AI Birds

When the game starts if the player's turn is in AI mode then the AI will automatically make the move. If the player is in human mode then the player has to input the next move.

- If it is Human Larva's turn, the player can press where the next move of the larva is. If the move is legal then the Larva changes its move.
- If it is Human Bird's turn, the player first has to press on the bird he/she wants to move. A black square will be drawn around the selected bird. The player can change the selected bird simply by pressing on another bird, and the black square will be highlighting the new selected bird. After the user chooses the bird he/she wants to move, the user can press where the next move of the bird is, if the move is legal then the selected bird changes its move.

If any move entered by the user is illegal, then nothing will happen and the user will still have to enter a legal move for player.

Programming Language

The programming language used for developing this game is C++. For the GUI, Juce library was used; a powerful, robust and cross-platform library. (http://www.juce.com/)

AI Algorithm

The algorithm used for the AI move is Minimax.

Data Structure

There are two modes for the minimax. If the game's turn is Larva's, then starting level of the minimax will be maximum level, else it will be a minimum level.

The data structure used to build the minimax is an array. All nodes of the minimax are saved in an array. In each mode there is a different number of total maximum nodes in the minimax at a constant minimax level. When the minimax is first created two arrays are defined, one for Max minimax and the other for Min minimax, each array has size of the maximum size of nodes possible in the minimax at the mode.

```
nodes[MAX] = new Game*[NumberOfNodes[MAX]];      //if larva's turn

nodes[MIN] = new Game*[NumberOfNodes[MIN]];      //if bird's turn
```

This design works only if the game defines a constant maximum possible number of moves for every player. The minimax defines that as:

```
NumberOfPossibleMoves[MAX] = 4;      // larva: up left or right, down left or right

NumberOfPossibleMoves[MIN] = 8;      // birds: up left or right for each bird
```

The maximum number of nodes in a mode is calculated as follow (turn is either MAX or MIN):

```
__int64 sum = NumberOfPossibleMoves[turn];   //first level is 4 or 8

__int64 LevelSum = sum;

For(int level = 1;  level < MaxLevel; level++)

{

        turn = ! turn; //to switch between max and min at every level

        LevelSum *= NumberOfPossibleMoves[turn]; // total nodes at this level

        Sum += levelSum;

}
```

Another four arrays are defined, two for each mode of the minimax.

```
levelBegins[MAX] = new __int64[maxlevel];
```

```
levelBegins[MIN] = new __int64[maxlevel];

nodesInLevel[MAX] = new __int64[maxlevel];

nodesInLevel[MIN] = new __int64[maxlevel];
```

The levelBegins array saves the index of the first node of each level in the nodes array.

And the nodesInLevel array saves the total number of nodes in this level.

So to get the nodes of level k at a specified mode (turn is MAX or MIN), the starting node will be at:

```
levelBegins[turn][k]
```

and the end node in this level will be at:

```
levelBegins[turn][k] + nodesInLevel[turn][k]
```

Creating the Nodes

To create the nodes in a level, the minimax uses a copy constructor of the Game class the takes a parameter of the player chosen to move and the type of move.

For example,

```
new Game(game, PlayerLarva, UpRight, true);
```

will create a game with same player positions of the passed game object but moves the Larva up to the right. If the move is an illegal move the game is flagged as invalid and then the node is deleted from the minimax array and the game is replaced with a nullptr.

The minimax starts by creating the first level game nodes, which are 4 games if in MAX mode or 8 nodes if in MIN mode.

Then the minimax starts creating the games at every level of the minimax, simply by getting the game parent and creating 4 other games using the parent game if MAX mode or 8 if MIN mode.

For example if the level is in MAX mode the minimax creates the following nodes for every node in the parent level.

```
nodes[isMax][i] = new Game(*getParent(i, level), PlayerLarva, UpRight, true);

nodes[isMax][i + 1] = new Game(*getParent(i + 1, level), PlayerLarva, UpLeft, true);

nodes[isMax][i + 2] = new Game(*getParent(i + 2, level), PlayerLarva, DownRight,true);

nodes[isMax][i + 3] = new Game(*getParent(i + 3, level), PlayerLarva, DownLeft,true);
```

Getting the Parent Node

To get the parent node of a specific node in the minimax:

```
__int64 parent = (levelBegins[isMax][LevelofNode] +

(node - levelBegins[isMax][LevelofNode]) /
NumberOfPossibleMoves[LevelofNodeisMax]

- nodesInLevel[isMax][LevelofNode - 1]);
```

For example, the parent node of node 12 in a MAX minimax mode:

Node = 12

levelBegins[MAX][1] = 4

NumberOfPossibleMoves[MIN] = 8

nodesInLevel[MAX][0] = 4

4 + (12-4)/8 – 4 = 1

So the parent of node at index 12 is node at index 1 in the nodes array.

Searching

The minimax starts at the last level of the minimax and checks every node in that level.
If the node and its parents are not 'nullptr' then it calculates the node's heuristic and pass it to the parent. If the parent already has a heuristic (passed to it from another child), the minimax checks if the new heuristic is bigger than or smaller than the existing heuristic and it either swaps or skip depending on the level of the node, and whether the level is maximizing or minimizing the heuristic.
The minimax continues passing the heuristic from every level to the parent level until it reaches the first level. Then it finds the best move based on the index of the node with the best heuristic at the first level.
For example, if it is Larva's turn then node at index 0 is Up Right and index 2 is Down Right and so on.

The complexity of the searching is O(k). The searching will go through all the nodes but most of the nodes will be 'nullptr' so no calculating heuristic or passing heuristic occurs.

With the 3 seconds constraint the minimax was able to go to level 8. However, there are multiple ways to increase the efficiency of the minimax which will allow me to go deeper than level 8.
First is by using alpha beta pruning. Second after finding the best move the minimax is deleting all the nodes in the array, which can be done in a separate thread.
Also, with the data structure used the searching itself can be done in multiple threads, for example nodes at each level will be divided into parts where every thread will take the nodes in this part and assign the heuristics to the parent of each node. The minimax will not go to the other level until all threads in the child level are done.

Heuristic

The method used to find the heuristic was trial and error. Some features I tried, I had high expectations for its result but ended up to be bad features, and vice versa.

The features used in the heuristic:

1. **Position of players on the Y axis.**

   For the larva:

   Heuristic + = (1 + players[PlayerLarva].PosY)*(8 - players[PlayerLarva].PosY + 1);

   And for the each bird:

      Only if the bird is below the larva then :

      Heuristic - = (1 + players[i].PosY)*(8 - players[i].PosY + 1);

   The one is for smoothing for when PosY = 0.

   The position is multiplied by (8- posY +1) to give a difference between a lower bird and a higher bird. For example:

0								
1								
2			L					
3								
4			B1					
5		B1						
6			B2					
7		B2		B3		B4		

   If the birds are in these positions, then the heuristic of birds positions when moving B1 up right or moving B2 up right without multiplying would be:

   B1 up = -(5+7+7+7) = -25

   B2 up = -(5+6+7+7) = -25

   Both moves will have the same heuristic considering birds position.

   However, when multiplying by (8-posY +1):

   B1 up = -(4*5+ 7*2+7*2+7*2) = - 62

   B2 up = (5*4 + 6*3 + 7*2 +7*2) = -66

   So moving B2 will be the better choice for the birds.

2. **Counting birds with respect to larva's Y position**

   If birds below larva is more or equal to 3 then

   heuristic -= 100

   else if birds below larva are less than or equal to 2 then
   heuristic += 100

3. **Birds at the same line**

   For all birds

   If a bird.PosY == thisBird.PosY then count++

   Where thisBird != to bird

   If all birds in one line the value of count will be 12.

   For the heuristic, if countBirdsSameLine>=9 then

   Heuristic -= 100 //only when there is at least 3 birds in line

4. **Using 2 features, birds below larva and birds at the same line**

   If birds below larva is 4 and count of birds at the same line is 12

   Then heuristic -=200

   This feature was not very affective; I should have combined this in feature 3. So feature 3 will only count birds on the same line if they are below the larva.

5. **Win or Lose**
   If larva wins then heuristic = 10000
   If Larva loses then heuristic = -10000

Competition

The competition went as expected. 3.5/4 games were won. I had the chance to ask every player I competed against about their heuristic. The opponent that won a battle against me had a very similar heuristic to mine which was interesting to watch as both our bird's behaved similarly against the larva and they both lost.

Another thing I noticed, was the importance of considering the level of the minimax as a feature to the heuristic.

For example, in some cases where the larva needs one move to win, the AI choses to play 7 moves to win instead. As both moves gives the same heuristic which is Win the searching is basically taking the nearest heuristic to the left of the tree in the minimax.

This can be corrected by adding a weight to the win or lose feature and multiplying its heuristic by (1/level).

For instance, if the node at level 2 wins the game and a node at level 4 wins the game then the heuristics will be

At 2 = 10000*(1/2) = 5000

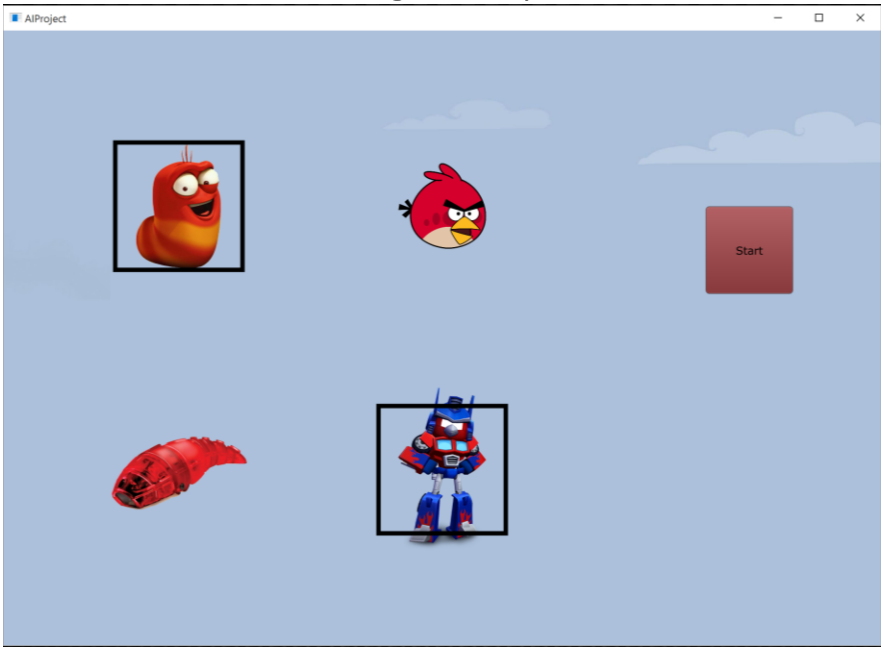
At 4 = 10000*(1/4) = 2500

And the larva will choose to win in 2 moves instead of 4.

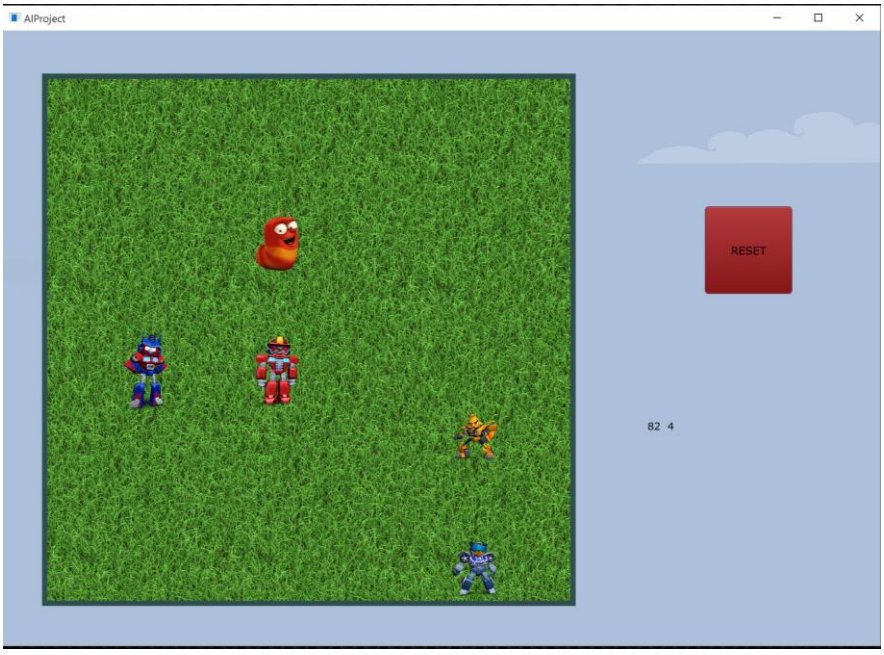
Without having this feature, the winning is not affected however the game looks stupid a bit for humans watching that will not understand why the AI chose not to win by one move instead took it multiple moves to win.

Screenshots

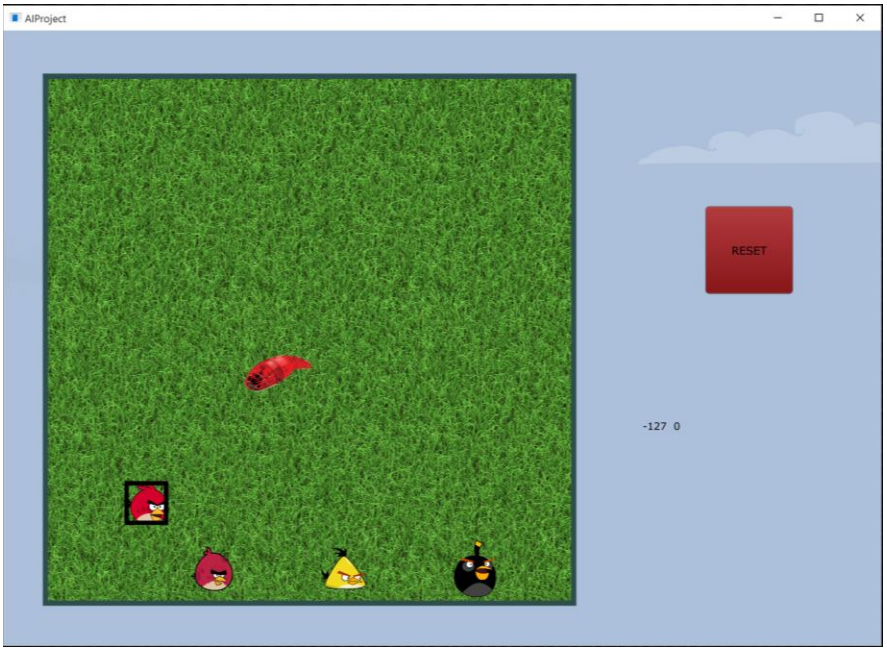
Main Menu and Choosing the Players



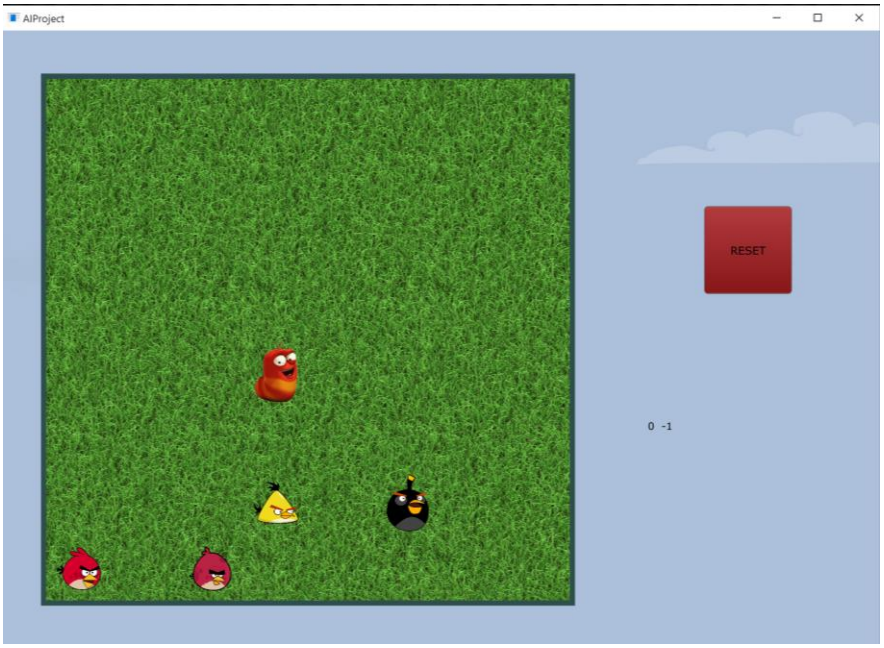
Human Larva VS AI Birds



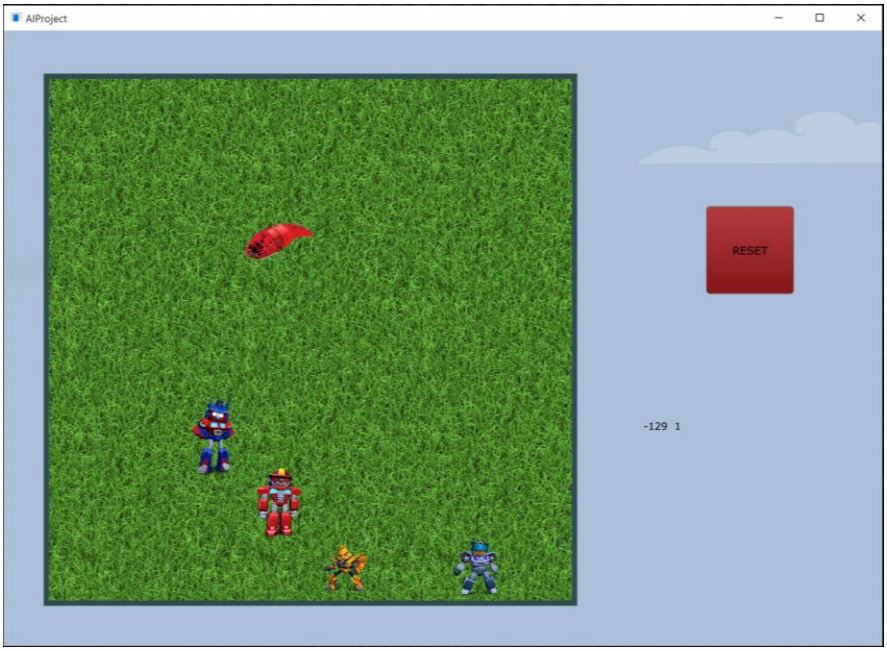
AI Larva VS Human Birds



Human Larva VS Human Birds



AI Larva VS AI Birds



Note: I do not own the rights for any of the graphics used.