Detailed Report:

Active Time Battle-Based Strategy Game Project

*by Tran Duy Duc*

# Introduction

This game combines Active Time Battle as seen in games such as *Final Fantasy VI* and strategy found in *Final Fantasy Tactics Advance.* It uses AutoIt scripting language, which can be easily used for GUI, simple graphics, easy array copying, and dynamic array resizing operations.

# In-Game Information

## Map

The map consists of a 6×6 board with no terrain. Each team, consisting of 3 members, is located at either side of the board. The left *(enemy)* team can select the initial location for characters within the first two columns of the board. Likewise, the right *(ally)* team can select within the last two columns of the board. The position of a character is unique; no squares are occupied by more than one character.



Figure 1. The Battle Scene window, consisting of the map

## Method of Determining Distance

The distance between two squares is calculated upon the Manhattan distance formula: with being the distance from the point with coordinates to the point

## Character Properties

Properties of a character include the following:

* *Hit Points:* Determines how much damage a character can sustain. Characters with 0 HP are considered knocked out and are removed from the map for the remainder of the game.
* *Magic Points:* Determines how many special abilities a character can cast until the end of the game. However, special abilities are temporarily removed for simplicity; therefore, this property is unused.
* *Attack:* Determines the amount of damage a character can deal to the target.
* *Defense:* Determines the amount of damage to be reduced. The incoming damage multiplier is calculated based on the formula from *League of Legends:*
* *Attack Range:* Determines the maximum distance from the character’s current position to an attack target.
* *Attack Area:* Determines the maximum distance from the center square where the attack is issued to the attack targets.
* *Speed:* Determines how fast the ATB bar is filled.
* *Movement Range:* Determines the maximum distance that a character can move during his turn.

There are in-game properties in addition to the preceding ones:

* *Time:* ATB bar filling progress.
* *X:* Abscissa of the character *(i.e. the column he is standing)*.
* *Y:* Ordinate of the character *(i.e. the row he is standing)*.

## Character Classes

There are five character classes with distinct playing styles:

* *Fighter:* Weak attacking power, sturdy defense.
* *Black Belt:* Strong attacking power.
* *Archer:* Long attacking range, poor defense.
* *Assassin:* Long movement range, extremely poor defense, and high speed for better mobility.
* *Bomber:* Able to damage surrounding enemies, poor defense.

The following table is the detailed properties of each class:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Class | HP | Atk | Def | Rng | Area | Spd | Mov |
| Fighter | 500 | 30 | 45 | 1 | 0 | 40 | 2 |
| Black Belt | 400 | 70 | 30 | 1 | 0 | 25 | 2 |
| Archer | 300 | 30 | 15 | 3 | 0 | 35 | 2 |
| Assassin | 250 | 45 | 15 | 1 | 0 | 80 | 3 |
| Bomber | 300 | 30 | 15 | 2 | 1 | 30 | 2 |

## Active Time Battle

Before the game starts, characters’ classes and initial positions are chosen. Class lineup must be the same for both teams, but members’ initial position can be freely chosen within the limit *(see* Map *subsection)*.

The time always passes in the game unless it is paused. If you act too slowly, the enemy may easily overwhelm you.

Every character has an individual ATB bar. When the bar is full, the corresponding character can take some actions before ending his turn, passing the turn to another character.

Up to two actions can be done per turn, in any order: *(1)* move, and *(2)* attack or ability *(removed for simplicity)*.

* *Move:* Moves a character within his Movement Range *(see* Character Properties *subsection)*. This action must ensure position uniqueness.
* *Attack:* Attacks an enemy. More than one enemy can take damage if they are within the attacker’s Attack Area *(see* Character Properties *subsection)*.

A character might end the turn with only one action or zero actions taken. His ATB bar will be partially filled, depending on the number of actions taken *(i.e. the fewer actions he takes, the sooner he will take his turn)*. If a character has his ATB bar full when someone else’s turn is being taken, he will be queued to be the next character to take a turn.

## Game Ending Conditions

That at least one team is completely annihilated *(i.e. all team members have 0 HP)* will cause the game to end*.* The team with at least one member alive wins.

If no attack command is issued in a turn, the turn is considered *“inactive.”* Thirty consecutive inactive turns *(i.e. 30 consecutive turns without anyone attacking)* will terminate the game. The team with more surviving characters wins. If number of survivors of both teams is the same, the team with higher total HP of members wins. Equal number of survivors and equal total HP of members will lead to a drawn game.

A drawn game can also occur when all characters have 0 HP; however, it typically cannot happen because turns are taken sequentially, not simultaneously.

## Status Window

In the status window, the player can read the battle log (primarily for reading the damage dealt by a character, also used for reading character behaviors), the remaining HP, MP, and ATB progresses. When a team member gets a turn, the player can issue commands in the same window. Number of inactive turns *(see* Game Ending Conditions *subsection)* can also be read so that the player can make his decisions more strategically.

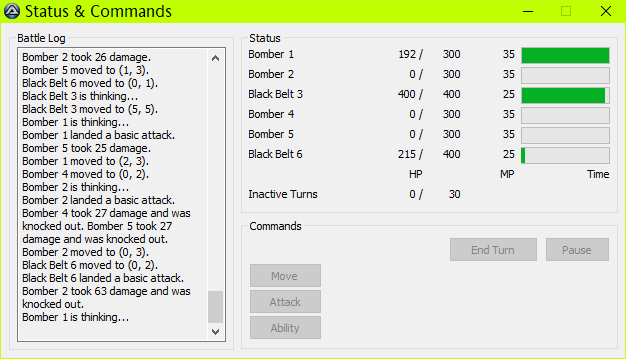


Figure 2. Status & Commands window

## Variables and Functions

### Map Size

Synopsis: Const MapSize = 6

This constant indicates the number of squares in a dimension of the map.

### Attacked

This is a Boolean variable used to indicate whether the character has attacked during his turn.

### Moved

This is a Boolean variable used to indicate whether the character has moved during his turn.

### Attack Character

Synopsis: AttackCharacter(DestinationX, DestinationY)

This function indicates an attack command with the center of the attack area on and sets the variable to True.

### Move Character

Synopsis: MoveCharacter(DestinationX, DestinationY)

This function indicates a move command with the destination on and sets the variable to True.

### Time Passes

Synopsis: TimePasses()

This function handles the turns and passing of time in Active Time Battle.

### Skip to Next Character

Synopsis: SkipToNext()

This function fast-forwards the time until a new character takes his turn. In this case, the function returns True. It returns False if the game ends *(at least one team is completely annihilated)*.

### Opposing Teams

Synopsis: OpposingTeams(Player1, Player2)

This function returns a Boolean value indicating whether two characters are in different teams.

### Backup States

Synopsis: BackupStates()

This function copies the current characters and turn queue into separate variables to start simulating next moves.

### Restore States

Synopsis: RestoreStates()

This function restores the original characters and turn queue when simulations are completed.

### Annihilated

Synopses:

* Annihilated(Allies)
* Annihilated(Enemies)

This function returns a Boolean value indicates whether the parameterized team is completely annihilated *(i.e. all members have 0 HP)*.

### Distance

Synopses:

* Distance((X1, Y1), (X2, Y2))
* Distance(Point1, Point2)

This function calculates the distance between two points using the Manhattan distance formula *(see* Method of Determining Distance *subsection)*.

### Random Integer

Synopsis: RandomInt(Min, Max)

This function returns a random integer between and inclusive.

### Random Float Number

Synopsis: RandomFloat(Min, Max)

This function returns a random float number between inclusive and exclusive *(i.e. )*.

# Artificial Intelligence of the Computer Player

## Random Moves (AI 1)

### Description

The player randomly attacks an enemy in range with 75% chance at the beginning of the turn. Then, it tries up to 10 random legal destinations to move *(i.e. within the Movement Range and not occupied by anyone)*. After 10 attempts, if it cannot find a legal move, it will not move. Finally, it attacks an enemy in range if it did not attack beforehand.

### Pseudocode

Func AIPlaysTheGame()

If (Not Attacked) And RandomInt(0, 4) <> 0 Then

For Each Enemy

If (Enemy HP > 0) And (Distance(Enemy, Player) <= Player Range) Then

AttackCharacter(Enemy X, Enemy Y)

ExitLoop

EndIf

Next

EndIf

If Not Moved Then

For I := 1 To 10

Destination X := RandomInt(

Max(0, Player X - Player Mov),

Min(MapSize - 1, Player X + Player Mov))

Destination Y := RandomInt(

Max(0, Player Y - Player Mov),

Min(MapSize - 1, Player Y + Player Mov))

LegalMove := True

For Each Character

If (Destination X = Character X) And

(Destination Y = Character Y) Then LegalMove := False

Next

If (Distance(Player, Destination) <= Player Mov) And LegalMove Then

MoveCharacter(Destination X, Destination Y)

ExitLoop

EndIf

Next

EndIf

If Not Attacked Then

For Each Enemy

If (Enemy HP > 0) And (Distance(Enemy, Player) <= Player Range) Then

AttackCharacter(Enemy X, Enemy Y)

ExitLoop

EndIf

Next

EndIf

EndFunc

### Comments

This is the worst and most basic AI. It is the base for Monte Carlo AI, which predicts the outcome of the game using random moves.

## Offensive (AI 2)

### Description

The player tries the lowest HP enemy in range. If the target is undefined or is knocked out, it will change the target for approaching; otherwise, it will randomly change the target with 10% chance. After that, the player approaches the enemy, and then attacks the lowest HP enemy in range if it did not attack beforehand.

For targeting, every distance unit is exchanged to 30 HP, ensuring that the player pursue the nearest and lowest HP enemy.

### Pseudocode

Func AIPlaysTheGame()

Const Weight = 30

If Not Attacked Then

For Each Enemy

If (Enemy HP <= 0) Or (Distance(Enemy, Player) > Player Range) Then

AISearch[Enemy] := 1E9

Else

AISearch[Enemy] := (Distance(Enemy, Player) \* Weight) + Enemy HP

EndIf

Next

LegalMove := False

For Each Enemy

If AISearch[Enemy] <> 1E9 Then LegalMove := True

Next

EndIf

If LegalMove And (Not Attacked) Then

AITarget := ArrayMinIndex(AISearch)

AttackCharacter(AITarget X, AITarget Y)

ElseIf (AITarget = -1) Or (AITarget HP <= 0) Or

RandomFloat(0, 1) < .1 Then

For Each Enemy

If Enemy HP <= 0 Then

AISearch[Enemy] := 1E9

Else

AISearch[Enemy] := (Distance(Enemy, Player) \* Weight) + Enemy HP

EndIf

Next

AITarget := ArrayMinIndex(AISearch)

EndIf

If Not Moved Then

For I := Max(0, Player X - Player Mov)

To Min(MapSize - 1, Player X + Player Mov)

For J := Max(0, Player Y - Player Mov)

To Min(MapSize - 1, Player Y + Player Mov)

LegalMove := True

For Each Character

If (Character X = I) And

(Character Y = J) Then LegalMove := False

Next

If LegalMove And (Distance((I, J), Player) <= Player Mov) Then

Region[I][J] := Distance((I, J), AITarget)

Else

Region[I][J] := 1E9

EndIf

Next

Next

Destination X := Max(0, Player X - Player Mov)

Destination Y := Max(0, Player Y - Player Mov)

For I := Max(0, Player X - Player Mov)

To Min(MapSize - 1, Player X + Player Mov)

For J := Max(0, Player Y - Player Mov)

To Min(MapSize - 1, Player Y + Player Mov)

If Region[I][J] < Region[Destination X][Destination Y] Then

Destination X := I

Destination Y := J

EndIf

Next

Next

If Region[Destination X][Destination Y] <> 1E9 Then

MoveCharacter(Destination X, Destination Y)

EndIf

EndIf

If Not Attacked Then

For Each Enemy

If (Enemy HP <= 0) Or (Distance(Enemy, Player) > Player Range) Then

AISearch[Enemy] := 1E9

Else

AISearch[Enemy] := (Distance(Enemy, Player) \* Weight) + Enemy HP

EndIf

Next

LegalMove := False

For Each Enemy

If AISearch[Enemy] <> 1E9 Then LegalMove := True

Next

If LegalMove Then

AITarget := ArrayMinIndex(AISearch)

AttackCharacter(AITarget X, AITarget Y)

EndIf

EndIf

EndFunc

### Comments

Though simple in design, this AI is efficient, making it a good trade-off between speed and intelligence. It was the best performing player before the re-programming of the updated AI 4.

## Ninja (AI 3)

### Description

The player tries to attack an enemy in range so that the number of enemies taking damage is as large as possible. If the attack is successful, it will try to run away from the enemies; otherwise, it approaches an enemy with the same rule as AI 2 and then attacks with the previous rule.

### Pseudocode

Func AIPlaysTheGame()

MaxX := Max(0, Player X - Player Rng)

MaxY := Max(0, Player Y - Player Rng)

If Not Attacked Then

For I := 0 To MapSize - 1

For J := 0 To MapSize - 1

Region[I][J] := 0

Next

Next

For I := Max(0, Player X - Player Rng)

To Min(MapSize - 1, Player X + Player Rng)

For J := Max(0, Player Y - Player Rng)

To Min(MapSize - 1, Player Y + Player Rng)

For Each Enemy

If Distance(Enemy, (I, J)) <= Player Area Then Region[I][J] += 1

Next

If Region[I][J] > Region[MaxX][MaxY] Then

MaxX := I

MaxY := J

EndIf

Next

Next

EndIf

For I := 0 To MapSize - 1

For J := 0 To MapSize - 1

Region[I][J] := 0

Next

Next

If (Not Attacked) And (Region[MaxX][MaxY] > 0)

And (Distance((MaxX, MaxY), Player) <= Player Rng) Then

AttackCharacter(MaxX, MaxY)

EndIf

If (AITarget = -1) Or (AITarget HP <= 0) Or RandomFloat(0, 1) < .1 Then

For Each Enemy

If Enemy HP <= 0 Then

AISearch[Enemy] := 1E9

Else

AISearch[Enemy] := Distance(Enemy, Player) \* Weight + Enemy HP

EndIf

Next

AITarget := ArrayMinIndex(AISearch)

EndIf

If Not Moved Then

If Attacked Then

MaxX := Max(0, Player X - Player Mov)

MaxY := Max(0, Player Y - Player Mov)

For I := Max(0, Player X - Player Mov)

To Min(MapSize - 1, Player X + Player Mov)

For J := Max(0, Player Y - Player Mov)

To Min(MapSize - 1, Player Y + Player Mov)

LegalMove := True

For Each Character

If (Character X = I) And (Character Y = J) Then

LegalMove := False

ExitLoop

EndIf

Next

If LegalMove Then

For Each Enemy

If Distance(Enemy, (I, J)) <= 3 Then Region[I][J] -= 1

Next

If Region[I][J] > Region[MaxX][MaxY] Then

MaxX := I

MaxY := J

EndIf

Else

Region[I][J] := -1E9

EndIf

Next

Next

For I := Max(0, Player X - Player Mov)

To Min(MapSize - 1, Player X + Player Mov)

For J := Max(0, Player Y - Player Mov)

To Min(MapSize - 1, Player Y + Player Mov)

If Region[I][J] = Region[MaxX][MaxY] Then

Moves[New Index] X := I

Moves[New Index] Y := J

EndIf

Next

Next

MaxIndex := RandomInt(0, NumberOfIndexes(Moves))

MoveCharacter(Moves[MaxIndex] X, Moves[MaxIndex] Y)

Else

For I := 0 To MapSize - 1

For J := 0 To MapSize - 1

LegalMove := True

For Each Character

If (Character X = I) And (Character Y = J) Then

LegalMove := False

ExitLoop

EndIf

Next

If LegalMove And (Distance((I, J), Player) <= Player Mov) Then

Region[I][J] := Distance((I, J), AITarget)

Else

Region[I][J] := 1E9

EndIf

Next

Next

Destination X := 0

Destination Y := 0

For I := 0 To MapSize - 1

For J := 0 To MapSize - 1

If Region[I][J] < Region[Destination X][Destination Y] Then

Destination X := I

Destination Y := J

EndIf

Next

Next

If Region[Destination X][Destination Y] <> 1E9 Then

MoveCharacter(Destination X, Destination Y)

EndIf

EndIf

EndIf

For I := 0 To MapSize - 1

For J := 0 To MapSize - 1

Region[I][J] := 0

Next

Next

If Not Attacked Then

For I := Max(0, Player X - Player Rng)

To Min(MapSize - 1, Player X + Player Rng)

For J := Max(0, Player Y - Player Rng)

To Min(MapSize - 1, Player Y + Player Rng)

For Each Enemy

If Distance(Enemy, (I, J)) <= Player Area Then Region[I][J] += 1

Next

If Region[I][J] > Region[MaxX][MaxY] Then

MaxX := I

MaxY := J

EndIf

Next

Next

If Distance((MaxX, MaxY), Player) <= Player Rng Then

AttackCharacter(MaxX, MaxY)

EndIf

EndIf

EndFunc

### Comments

This player paradoxically performs worse than AI 2, although it tries to run away from the enemies after attacking. There is little chance that this AI wins when playing against AI 2. Its retreating moves are silly sometimes.

## Monte Carlo 1 (AI 4)

### Description

The player lists possible moves by each phase *(attack, move, and attack again if possible)*; all moves are listed for attack phase and some are listed for move phase with the following rule.

For each move, 50 games are simulated with random moves *(using AI 1)* for up to 8 turns. The score for a simulation has HP of all characters taken into account, is accumulated through simulations, and is calculated using the following formula:

Ending Factor equals 2 if at least one team is completely annihilated and 1 otherwise. It intensifies the need of ending the game. Likewise, number of surviving enemies emphasizes the significance of knocking out a single-digit HP enemy.

The move with the highest final score will be chosen.

### Pseudocode

Func AIPlaysTheGame() *// Shared for AI modes 4, 5, and 6*

BackupStates()

AIMoves[0] Action := Do Nothing

AIMoves[0] Score := 0

For I := Max(0, Player X - Player Rng)

To Min(MapSize - 1, Player X + Player Rng)

For J := Max(0, Player Y - Player Rng)

To Min(MapSize - 1, Player Y + Player Rng)

If Distance((I, J), Player) <= Player Rng Then

For Each Enemy

If (Enemy HP > 0) And

(Distance(Enemy, (I, J)) <= Player Area) Then

AIMoves[New Index] Action := Attack

AIMoves[New Index] X := I

AIMoves[New Index] Y := J

ExitLoop

EndIf

Next

EndIf

Next

Next

Testing()

BackupStates()

ClearArray(AIMoves)

AIMoves[0] Action := Do Nothing

AIMoves[0] Score := 0

Found := False

For I := Max(0, Player X - Player Mov)

To Min(MapSize - 1, Player X + Player Mov)

For J := Max(0, Player Y - Player Mov)

To Min(MapSize - 1, Player Y + Player Mov)

LegalMove := True

For Each Character

If (Character X = I) And (Character Y = J) Then LegalMove := False

Next

If LegalMove And (Distance((I, J), Player) <= Player Mov) Then

For Each Enemy

If (Enemy HP > 0) And

(Distance((I, J), Enemy) <= Player Rng) Then

Found := True

AIMoves[New Index] Action := Move

AIMoves[New Index] X := I

AIMoves[New Index] Y := J

ExitLoop

EndIf

Next

EndIf

Next

Next

If Not Found Then

For I := 1 To 6

Destination X := RandomInt(Max(0, Player X - Player Mov),

Min(MapSize - 1, Player X + Player Mov))

Destination Y := RandomInt(Max(0, Player Y - Player Mov),

Min(MapSize - 1, Player Y + Player Mov))

LegalMove := True

For Each Character

If (Character X = Destination X) And (Character Y = Destination Y)

Then LegalMove := False

Next

If LegalMove And (Distance(Destination, Player) <= Player Mov) Then

AIMoves[New Index] Action := Move

AIMoves[New Index] X := Destination X

AIMoves[New Index] Y := Destination Y

EndIf

Next

EndIf

Testing()

BackupStates()

ClearArray(AIMoves)

AIMoves[0] Action := Do Nothing

AIMoves[0] Score := 0

If Not Attacked Then

For I := Max(0, Player X - Player Rng)

To Min(MapSize - 1, Player X + Player Rng)

For J := Max(0, Player Y - Player Rng)

To Min(MapSize - 1, Player Y + Player Rng)

If Distance((I, J), Player) <= Player Rng Then

For Each Enemy

If (Enemy HP > 0) And

(Distance(Enemy, (I, J)) <= Player Area) Then

AIMoves[New Index] Action := Attack

AIMoves[New Index] X := I

AIMoves[New Index] Y := J

ExitLoop

EndIf

Next

EndIf

Next

Next

Testing()

EndIf

RestoreStates()

EndFunc

Func Testing()

AIMode := AI 1

For I In AIMoves

For K := 1 To 50

CurrentScore := 0

TestingTurns := 0

EndingFactor := 1

Switch AIMoves[I] Action

Case Move

MoveCharacter(AIMoves[I] X, AIMoves[I] Y)

Case Attack

AttackCharacter(AIMoves[I] X, AIMoves[I] Y)

EndSwitch

AIPlaysTheGame()

While TestingTurns < 8

If Annihilated(Allies) Or Annihilated(Enemies) Then

EndingFactor := 2

ExitLoop

EndIf

TimePasses()

WEnd

Survivors := 0

For Each Ally

If Ally HP > 0 Then Survivors += 1

Next

For Each Ally

CurrentScore += Ally HP \* Survivors

Next

Survivors := 0

For Each Enemy

If Enemy HP > 0 Then Survivors += 1

Next

For Each Enemy

CurrentScore -= Enemy HP \* Survivors

Next

AIMoves[I] Score += CurrentScore \* EndingFactor

RestoreStates()

Next

Next

AIMode := AI 4

MaxScore := AIMoves[0] Score

For I In AIMoves

If AIMoves[I] Score > MaxScore Then MaxScore := AIMoves[I] Score

Next

For I In AIMoves

If AIMoves[I] Score = MaxScore Then MaxIndexes[New Index] := I

Next

MaxIndex := MaxIndexes[RandomInt(0, NumberOfIndexes(MaxIndexes))]

RestoreStates()

Switch AIMoves[MaxIndex] Action

Case Move

MoveCharacter(AIMoves[MaxIndex] X, AIMoves[MaxIndex] Y)

Case Attack

AttackCharacter(AIMoves[MaxIndex] X, AIMoves[MaxIndex] Y)

EndSwitch

Return AIMoves[MaxIndex] Action

EndFunc

### Comments

This player makes its move slowly. On a PC running Windows 10 with an Intel Core i5-4200U CPU, it takes typically about 10 seconds to complete its turn. Advanced at the expense of speed, it is the best performing AI so far.

## Minimax (AI 5)

### Description

The player lists all of the possible moves by each phase *(attack, move, and attack again if possible)*. For each move, AI 2 is used to simulate all moves of the next character to take a turn. The score is evaluated using the scoring formula from AI 4, and its sign is changed if the character whose score is evaluated is an enemy. The player ultimately chooses the move that delivers the highest score.

### Pseudocode

Func AIPlaysTheGame() *// Uses the same function from AI 4*

Func Testing() *// Shared for AI 5 and AI 6*

AIMode := AI 2

For I In AIMoves

If Original AIMode = AI 6 Then GrandTotal := 0

AIMoves[I] Score := HierarchicalTesting(I, 3)

RestoreStates()

Next

AIMode := Original AIMode

MaxScore := AIMoves[0] Score

For I In AIMoves

If AIMoves[I] Score > MaxScore Then MaxScore := AIMoves[I] Score

Next

For I In AIMoves

If AIMoves[I] Score = MaxScore Then MaxIndexes[New Index] := I

Next

MaxIndex := MaxIndexes[RandomInt(0, NumberOfIndexes(MaxIndexes))]

RestoreStates()

Switch AIMoves[MaxIndex] Action

Case Move

MoveCharacter(AIMoves[MaxIndex] X, AIMoves[MaxIndex] Y)

Case Attack

AttackCharacter(AIMoves[MaxIndex] X, AIMoves[MaxIndex] Y)

EndSwitch

Return AIMoves[MaxIndex] Action

EndFunc

Func HierarchicalTesting(NodeIndex, Depth, Alpha = -1E18, Beta = +1E18)

*// Shared for AI 5 and AI 6*

Const Simulations = 10

CurrentScore := 0

Total := 0

BackupStates()

Switch AIMoves[I] Action

Case Move

MoveCharacter(AIMoves[I] X, AIMoves[I] Y)

AIPlaysTheGame()

Case Attack

AttackCharacter(AIMoves[I] X, AIMoves[I] Y)

AIPlaysTheGame()

Case Else

CompleteTurn()

EndSwitch

If OriginalMode = AI 6 Then

TestingTurns := 0

While TestingTurns < 5

If Annihilated(Allies) Or Annihilated(Enemies) Then ExitLoop

TimePasses()

WEnd

EndIf

If (Depth <= 0) Or (Not SkipToNext()) Then

Survivors := 0

For Each Ally

If Ally HP > 0 Then Survivors += 1

Next

For Each Ally

CurrentScore += Ally HP \* Survivors

Next

Survivors := 0

For Each Enemy

If Enemy HP > 0 Then Survivors += 1

Next

For Each Enemy

CurrentScore -= Enemy HP \* Survivors

Next

If Annihilated(Allies) Or Annihilated(Enemies) Then CurrentScore \*= 2

If OpposingTeams(CurrentPlayer, TestingPlayer) Then CurrentScore \*= -1

RestoreStates()

Return CurrentScore

EndIf

CurrentScore := -1E18

Total := 0

BackupStates()

For Each Operation

*// Operations are listed in this order: Attack, Move, Do Nothing*

*// Processed the same way as AIPlaysTheGame()*

AIMoves[New Index] Action := Operation Action

AIMoves[New Index] X := Operation X

AIMoves[New Index] Y := Operation Y

AIMoves[New Index] Parent := NodeIndex

If OriginalMode = AI 5 Then

AIMoves[New Index] Score :=

OpposingTeams(CurrentPlayer, TestingPlayer) ?

HierarchicalTesting(New Index, Depth - 1, -Beta, -Alpha) :

HierarchicalTesting(New Index, Depth - 1, Alpha, Beta)

Else

AIMoves[New Index] Score := 0

For I := 1 To Simulations

AIMoves[New Index] Score += HierarchicalTesting(New Index, 0)

Next

Total += Simulations

EndIf

If AIMoves[New Index] Score > CurrentScore Then

CurrentScore := AIMoves[New Index] Score

EndIf

If AIMoves[New Index] Score > Alpha Then

Alpha := AIMoves[New Index] Score

EndIf

If Alpha >= Beta Then ExitLoop

RestoreStates()

Next

If OriginalMode = AI 6 Then

MaxIndex := First Child Of NodeIndex

For Each Child Of NodeIndex

AIMoves[Child] Score := (AIMoves[Child] Score / Simulations) +

Sqrt(100000 \* Ln(Total) / Simulations)

If AIMoves[Child] Score > AIMoves[MaxIndex] Score Then

MaxIndex := Child

EndIf

Next

GrandTotal += Total

CurrentScore := (HierarchicalTesting(MaxIndex, Depth - 1) /

Simulations) + Sqrt(100000 \* Ln(GrandTotal) / Simulations)

EndIf

RestoreStates()

Return CurrentScore

EndFunc

### Comments

The time needed for this AI to move varies greatly from seconds to minutes with depth 3; however, its efficiency does not surpass that of AI 2 much.

## Monte Carlo 2 (AI 6)

### Description

This AI player follows Monte Carlo Tree Search principle: it digs deeper in the node with the best score *(calculated using the formula from AI 4)*. After evaluating its descendants, the score is updated using the following formula:

The move giving the highest score will be chosen.

### Pseudocode

See section 3.5.2.

### Comments

The time needed for this player to complete its move is comparable to that of AI 4. However, it is not as intelligent as AI 4.

## AI Efficiency Comparison Results

For every game, a team has the random class lineup and the other team copies the same composition for fairness. No drawn games were counted in the total number of games *(i.e. only games with one winner were counted)*. With the AI player numbering conventions introduced from the preceding subsections, the results of a 1000-game sample for each AI type are as follows:

* AI 2 has 100% winning ratio against AI 1
* AI 3 has 7.5% winning ratio against AI 2
* AI 4 has 64.5% winning ratio against AI 2
* AI 5 has 54.9% winning ratio against AI 2
* AI 6 has 62% winning ratio against AI 2
* AI 6 has 29.7% winning ratio against AI 4

From these results, the difficulty ranking of the proposed AI players is as follows, from easiest to hardest:

* Random Moves (AI 1)
* Ninja (AI 3)
* Offensive (AI 2)
* Minimax (AI 5)
* Monte Carlo 2 (AI 6)
* Monte Carlo 1 (AI 4)

# Conclusion

AI 6 looks promising, but it might be suffering from mis-implementation. Further attention will be paid to this AI mode.

Machine learning is the future of this project. It will be used to improve the efficiency of AI players.

*Nomi, August 24, 2017*

*Tran Duy Duc*