Assignment 2: AI Arkanoid

Name: Pengcheng Jin
ID: 201447767

1. Decision trees description

Decision tree is a supervised learning algorithm (having a pre-defined target variable), which is mostly used in classification problems. As the earliest artificial intelligence algorithm, it has already developed many generations, such as AID, ID3, CART, C4.5 and C5.

Most important things when people choosing decision trees as decision-making algorithm is building an appropriate Tree to simulate it. To build a suitable tree, people should **Find Best Split, Determine Candidate Split and Stopping criteria**.

In the game design, more and more popular games were added some kind of AI player to increase playability, such as PUBG and LOL. Although these two examples above using more complex AI algorithm, many game situations can be described as **if-then-else cases**, which are decision tree.

In my view, decision tree could be divided three parts, which are **knowledge, classification and acting on knowledge**. Through classification to search stored knowledge then acting on knowledge. Classification is some if-then-else statements, which is the most important core in decision tree. As the simplest decision-making technique, decision tree is widely used in most of games.

2. An AI paddle Design description

This game has 3 rounds, the difficulty of round is gradual progression. There are two method to increase the difficult, which are increase red ball (fight ball) velocity and Ai paddle movement velocity. There are two formula to compute it.

Red ball (fight ball) velocity= base velocity (9) + elapsed time (time / 3) + number of user wins (UserScore) + decrease in obstacles (HitScore);

Ai paddle movement velocity= round base velocity (number of wins) + decrease in obstacles (HitScore)+VelocityDecisionTree (GoFast, GoSlow).

Red ball (fight ball) velocity are controlled in simpleUpdate(float tpf) method. Through repeatedly loop in this method, it could become a velocity change supervisor to adjust red ball velocity all the time.

```
640         //this is a fight ball velocity keeper
641         rigidBodyFight.setLinearVelocity(rigidBodyFight.getLinearVelocity().normalize().mult(9 + time / 3 + UserScore + HitScore ));
```

AI paddle design:

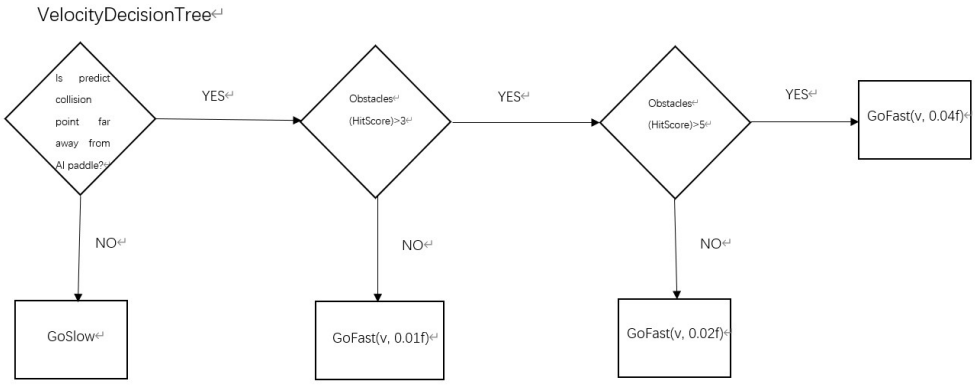
Ai paddle movement velocity are controlled in simpleUpdate(float tpf) and VelocityDecisionTree(float v) method. In order to explain Ai paddle movement velocity, Ai paddle movement principle should introduce first.

Ai paddle principle: Ai paddle movement are controlled in simpleUpdate(float tpf) method, so it means Ai paddle move step by step in every loop. simpleUpdate(float tpf) method will invoke VelocityDecisionTree and MoveDecisionTree in every loop, and then two Decision Trees will compute predict collision point location and compare it with Ai paddle location to decide whether move a little step.

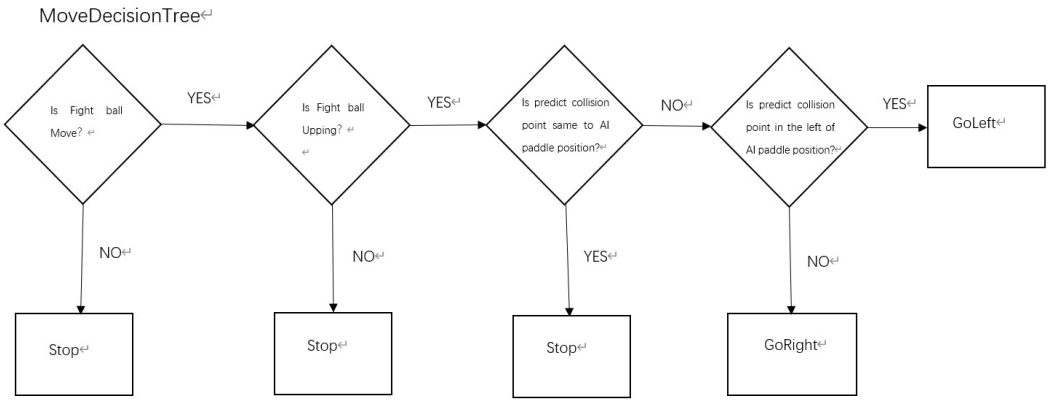
```
637         //next two lines are 2 decision trees
638         float AiPaddleVelocity = VelocityDecisionTree(0.05f*100*tpf);//use velocitydecisiontree to compute ai paddle move velocity
639         MoveDecisionTree(AiPaddleVelocity);//use MoveDecisionTree to choose left or right to move.
```

Return to Ai paddle movement velocity, it is through change step length to control movement velocity in every loop.

2.1 Graphical representation of the tree



VelocityDecisionTree are focus on the Ai paddle movement velocity by GoFast and GoSlow behaviors. The major decision conditions are the coordinates of the red ball and decrease in obstacles. The details of this decision tree will be discussed in next part.



MoveDecisionTree are focus on the Ai paddle moving by GoLeft, GoRight and Stop

behaviors. The major decision conditions are the velocity direction of the red ball, the coordinates of the red ball. The details of this decision tree will be discussed in next part.

2.2 Clearly indicate tests and actions

Tests:

VelocityDecisionTree					
Condition				Predict behavior	results
Red ball location	Red ball velocity	Ai paddle location	HitScore	x	x
(0,-4,0)	(0,10,0)	(0,8.2,0)	0	GoSlow	GoSlow
(0,2,0)	(4,4,0)	(0,8.2,0)	2	GoFast(v,0.01f)	GoFast(v,0.01f)
(1,0,0)	(4,2,0)	(0,8.2,0)	5	GoFast(v,0.02f)	GoFast(v,0.02f)
(3,1,0)	(-3,-3,0)	(0,8.2,0)	6	GoFast(v,0.04f)	GoFast(v,0.04f)

MoveDecisionTree				
Condition			Predict behavior	results
Red ball location	Red ball velocity	Ai paddle location	x	x
(0,-7.45,0)	(0,0,0)	(0,8.2,0)	Stop	Stop
(0,-3,0)	(1,-5,0)	(0,8.2,0)	Stop	Stop
(0,-7.45,0)	(0,8,0)	(0,8.2,0)	Stop	Stop
(-2,0,0)	(6,3,0)	(0,8.2,0)	GoRight	GoRight
(6,-2,0)	(-9,3,0)	(0,8.2,0)	GoLeft	GoLeft

Action:

User Paddle:

Left or **Right** is controlled by keyboard.

Ai Paddle:

GoLeft: if predict collision point is in the left of Ai paddle location.

GoRight: if predict collision point is in the right of Ai paddle location.

Stop: if predict collision point is same to Ai paddle location or red ball still or red ball is going to user paddle.

GoFast: if predict collision point is far away from Ai paddle location and decrease in obstacles (HitScore)

GoSlow: if predict collision point is close to Ai paddle location

2.3 Justify design decisions

At first in the VelocityDecisionTree, it can adjust AI paddle movement velocity all the time. It can avoid Ai paddle reaches the predicted point too early or too late. At the same time, it can increase the AI paddle move velocity with the decrease in obstacles to improve the difficulty of this game gradually.

Then in the MoveDecisionTree, is fight ball move and is fight ball upping two decision condition can avoid AI paddle unnecessary movement. Is predict collision point are same with AI paddle location decision condition can avoid AI paddle shake at a position (Flip-Flopping Decisions).

In order to increase the fun of this game, I decrease the round 1's difficult. In round 1, AI paddle velocity is very slow, it can be sure user win in first round. Then in the second round, AI paddle difficult is nearly a real player, so user should try their best to win. In round 3, AI paddle is nearly invincible. I called this structure is best of three sets, and this structure can be sure user have great chance to win this game.

All above of this are why my decision tree are a good AI model.

3. **A description of implementation**

This assignment using **hard code decision tree** to implement AI paddle movement. The details are as follows. It will be divided decision condition part, behavior part and decision tree main part in every decision tree.

VelocityDecisionTree:

Decision condition part:

```
727  public double CollisionLocationX() {
         double CollisionLocationX = 0;
729      rigidBodyFight.getLinearVelocity();
730      rigidBodyFight.getPhysicsLocation();
731      //algorithm
732      ///(CollisionLocationX-fightball location.x)/(CollisionLocationY - fightball location.y)=fight ball velocity.x / fightball location.y
733      //CollisionLocationY=8.2f
734      CollisionLocationX = ((-(rigidBodyFight.getLinearVelocity().x / rigidBodyFight.getLinearVelocity().y) * (-8.2 + rigidBodyFight.getPhysicsLocation().y)) + ri
735      return CollisionLocationX;
736  }
737
     public boolean PredictFarToPaddle() {
         boolean PredictFarToPaddle = true;
         if ((CollisionLocationX() - rigidBodyAiPaddle.getPhysicsLocation().x) > 1 || (CollisionLocationX() - rigidBodyAiPaddle.getPhysicsLocation().x) < -1) {
741          PredictFarToPaddle = true;
742      } else {
743          PredictFarToPaddle = false;
744      }
745      return PredictFarToPaddle;
746  }
```

CollisionLocationX() are the algorithm of compute predict collision location between red ball and AI paddle. At first CollisionLocation Vector3f.z is fixed, so do not consider Z axis. Then CollisionLocation.y is the same of Ai paddle.y, and the red ball velocity direction, location is known. So just figure out (predict collision location).x by vector computation.

PredictFarToPaddle(): Through get CollisionLocationX() and AI paddle location to test whether predict collision location are far away from Ai paddle location. If -1<distance<1, return false. if distance>1 or distance<-1, return true.

Behavior part:

```
public float GoFast(float v, float increment) {
    v = v + increment;
    return v;
}

/**
 * GoSlow behaviour, if predict location are not far away Ai paddle
 * location, GoSlow
 *
 * @param v
 * @return v = v - 0.01f;
 */
public float GoSlow(float v) {
    v = v - 0.01f;
    return v;
}
```

GoFast(float v, float increment): speed up AI paddle movement.

AI paddle velocity=v(initial velocity)+increment(decrease in obstacles)

GoSlow(float v): slow down AI paddle movement.

AI paddle velocity=v(initial velocity)-0.01f

Decision tree main part:

```
public float VelocityDecisionTree(float v) {
    if (PredictFarToPaddle() == true) {
        if (HitScore > 3) {
            if (HitScore > 5) {
                v = GoFast(v, 0.04f);
            } else {
                v = GoFast(v, 0.02f);
            }
        } else {
            v = GoFast(v, 0.01f);
        }
    } else {
        v = GoSlow(v);
    }
    return v;
}
```

If predict collision location are close to AI paddle location, need decrease AI paddle velocity rather than speed up.

If predict collision location are far away from AI paddle location, need increase AI paddle velocity. According to different HitScore(decrease in obstacles), the Ai paddle velocity increment are divided into 3 parts, which are +0.01f,+0.02f,+0.04f.

MoveDecisionTree:

Decision condition part:

```
    public boolean FightballMove() {
        boolean FightballMove = true;
        if (rigidBodyFight.getLinearVelocity().length() != 0) {
            FightballMove = true;
        } else if (rigidBodyFight.getLinearVelocity().length() == 0) {
            FightballMove = false;
        }
        return FightballMove;
    }
    public boolean FightballUp() {
        boolean FightballUp = true;
        if (rigidBodyFight.getLinearVelocity().y >= 0) {
            FightballUp = true;
        } else if (rigidBodyFight.getLinearVelocity().y < 0) {
            FightballUp = false;
        }
        return FightballUp;
    }
    public boolean SameCollision() {
        boolean SameCollision = true;
        if ((CollisionLocationX() - rigidBodyAiPaddle.getPhysicsLocation().x) > -0.2 && (CollisionLocationX() - rigidBodyAiPaddle.getPhysicsLocation().x) < 0.2)
            SameCollision = true;
        } else {
            SameCollision = false;
        }
        return SameCollision;
    }
    public boolean LeftCollision() {
        boolean LeftCollision = true;
        if (CollisionLocationX() < rigidBodyAiPaddle.getPhysicsLocation().x) {
            LeftCollision = true;
        } else if (CollisionLocationX() > rigidBodyAiPaddle.getPhysicsLocation().x) {
            LeftCollision = false;
        }
        return LeftCollision;
    }
```

FightballMove(): Through get red ball(fight ball) velocity direction to test whether red ball are move. If red ball is motionless, return false. if red ball is moving, return true.

FightballUp(): Through get red ball velocity direction to test whether red ball are upping, if upping, move ai paddle. If red ball is moving down, return false. if red ball is moving upper, return true.

SameCollision(): Through get CollisionLocationX() and AI paddle location to test whether predict location is same or very close to AI paddle. If predict collision location is not equal to AI paddle location, return false. if predict collision location is nearly equal to (-0.2~0.2) AI paddle location, return true. The reason why nearly equal is to avoid AI paddle shake and Flip-Flopping Decisions.

LeftCollision(): Through get CollisionLocationX() and AI paddle location to test whether predict collision location is in the left of Ai paddle. If predict collision location in the right of AI paddle location, return false. if predict collision location in the left of AI paddle location, return true.

Behavior part:

```
    public void GoLeft(float v) {
        if (rigidBodyAiPaddle.getPhysicsLocation().x > -11f) {// give AI paddle a left bound to move
            Vector3f left = new Vector3f(-v, 0f, 0f);
            rigidBodyAiPaddle.setPhysicsLocation(rigidBodyAiPaddle.getPhysicsLocation().add(left));
        }
    }
    public void GoRight(float v) {
        if (rigidBodyAiPaddle.getPhysicsLocation().x < 11f) {// give AI paddle a right bound to move
            Vector3f right = new Vector3f(v, 0f, 0f);
            rigidBodyAiPaddle.setPhysicsLocation(rigidBodyAiPaddle.getPhysicsLocation().add(right));
        }
    }
    public void Stop() {
        rigidBodyAiPaddle.getPhysicsLocation();
    }
```

GoLeft(float v): set AI paddle a new left location, and v is the step length.

GoRight(float v): set AI paddle a new right location, and v is the step length.

Stop(): do nothing.

Decision tree main part:

```
905     public void MoveDecisionTree(float v) {
906
907         if (FightballMove() == true) {
908             if (FightballUp() == true) {
909                 if (SameCollision() == true) {
910                     Stop();
911                 } else if (SameCollision() == false) {
912                     if (LeftCollision() == true) {
913                         GoLeft(v);
914                     } else if (LeftCollision() == false) {
915                         GoRight(v);
916                     }
917                 }
918             } else if (FightballUp() == false) {
919                 Stop();
920             }
921         } else if (FightballMove() == false) {
922             Stop();
923         }
924     }
925 }
926
```

Only if red ball is moving and upper, predict collision location are different from AI paddle location and predict collision location is in the left of AI paddle, GoLeft behavior will execute.

Only if red ball is moving and upper, predict collision location are different from AI paddle location and predict collision location in the right of AI paddle, GoRight behavior will execute.

All the others condition behaviors are Stop.