microRTS 初步文档

讨论组q：798578364

ai.core.?

AI

`所有AI`的基础抽象类，i.e，其他的AI定义都需继承自AI并实现以下四个方法

- 重置bot

```
public abstract void reset();
```

- ***核心代码部分，创建bot的接口***

```
public abstract PlayerAction getAction(int player, GameState gs) ;
```

- 返回bot的拷贝

```java
public abstract AI clone();
```

- 获取bot的参数

```java
public abstract List<ParameterSpecification> getParameters();
```

AIWithComputationBudget

继承AI，带有计算力限制的AI类，其执行受限于`计算时间`和`迭代次数`

AI.abstraction.?

AbstractionLayerAI

主要描述了AI的抽象动作

- 构造函数

```java
public AbstractionLayerAI(PathFinding a_pf) //接收寻路算法
public AbstractionLayerAI(PathFinding a_pf, int timebudget, int cyclesbudget)
```

- ***将抽象动作具体化***

```java
public PlayerAction translateActions(int player, GameState gs)
```

- 抽象动作方法：

```java
public void move(Unit u, int x, int y)
public void  train(Unit u, UnitType unit_type)
public void build(Unit u, UnitType unit_type, int x, int y)
public void harvest(Unit u, Unit target, Unit base)
public void attack(Unit u, Unit target)
public void idle(Unit u)
public int findBuildingPosition(List<Integer> reserved, int desiredX, int desiredY, Player p, PhysicalGameState pgs)
public boolean buildIfNotAlreadyBuilding(Unit u, UnitType type, int desiredX, int desiredY, List<Integer> reservedPositions, Player p, PhysicalGameState pgs)
```

rts.units.?

UnitType

广泛的对单位的定义

UnitTypeTable

使用UnitType进行构造游戏中单位的类型，含有单位的属性。存储在一个***列表***当中，决定了游戏的平衡性。

- 设置游戏中存在的单位

```
public void setUnitTypeTable(int version, int crs){//版本号，冲突处理法则
//e.g. RESOURCE:
UnitType resource = new UnitType();
resource.name = "Resource";
resource.isResource = true;
resource.isStockpile = false;
resource.canHarvest = false;
resource.canMove = false;
resource.canAttack = false;
resource.sightRadius = 0;
addUnitType(resource);
}
```

- 获取定义的单位类型

```
public UnitType getUnitType(int ID)
//Resource, Base, Barracks, Worker, Light, Heavy, Ranged
public UnitType getUnitType(String name)//e.g. getUnitType("Worker")
public List<UnitType> getUnitTypes()
```

Unit

描述`单位在游戏中`的信息

- 构造函数，接收单位id（可省略，自动增长），所属玩家，单位类型，所在坐标，携带资源数目

```
public Unit(long a_ID, int a_player, UnitType a_type, int a_x, int a_y,
int a_resources)
public Unit(Unit other) //用另一个单位来构造
```

- 对应于构造函数参数的get，set方法。
- 其他GS方法

```
public int getHitPoints()
public int getMaxHitPoints()
public void setHitPoints(int a_hitpoints)
public int getCost() //建造该单位的资源代价
public int getMoveTime()
public int getAttackTime()
public int getAttackRange()
public int getMinDamage()
public int getMaxDamage()
public int getHarvestAmount() //最大能够收获的资源数
```

- ***获取在某一游戏状态下的理论上合法的位置***

  1. 对未加分配的动作***默认处以10个cycle***的空操作
  2. 对未加分配的动作***自定义空动作的周期***

```
public List<UnitAction> getUnitActions(GameState s)
public List<UnitAction> getUnitActions(GameState s, int noneDuration)
```

- 判断一个单位ua，在给定状态gs下***是否有动作可以执行***

```
public boolean canExecuteAction(UnitAction ua, GameState gs)
```

- clone()

rts.?

UnitAction

描述了单位的动作信息

- 动作编码

```
public static final int TYPE_NONE = 0;//空操作
public static final int TYPE_MOVE = 1;//移动
public static final int TYPE_RETURN = 3;//带物回城
public static final int TYPE_PRODUCE = 4;//生产
public static final int TYPE_ATTACK_LOCATION = 5;//攻击
public static final int NUMBER_OF_ACTION_TYPES = 6;//操作数量
```

- 方向编码

```
public static final int DIRECTION_NONE = -1;//🖐
public static final int DIRECTION_UP = 0;   //👆
public static final int DIRECTION_RIGHT = 1;//👉
public static final int DIRECTION_DOWN = 2; //👇
public static final int DIRECTION_LEFT = 3; //👈
```

- 构造函数（不同函数对应不同的操作的构造法）

```
public UnitAction(int a_type)
public UnitAction(int a_type, int a_direction)
public UnitAction(int a_type, int a_direction, UnitType a_unit_type)//e.g.适用于produce(e.g. UnitAction(TYPE_PRODUCE, DIRECTION_DOWN, a_unit_type))
public UnitAction(int a_type, int a_x, int a_y)
public UnitAction(UnitAction other)
```

- ***估算单位执行动作的时间***

```
public int ETA(Unit u)
```

- 直接在游戏中将动作交由执行，`注意，这将在底层直接执行动作`

```
public void execute(Unit u, GameState s)
```

- 其他方法

```
public String getActionName()
public int getDirection()
public int getLocationX()
public int getLocationY()
```

Player

描述玩家的信息

- 构造函数，接收玩家id和分配给其的资源

```
public Player(int a_ID, int a_resources)
```

- 获取ID，资源，克隆

```
public int getID()
public int getResources()
public Player clone()
```

PlayerAction

存储各单位的动作，以二元组（Unit, UnitAction）**列表**存储。

- 判断是否为空

```
public boolean isEmpty()
```

- 判断玩家是否有任意一个不同于UnitAction.TYPE_NONE的动作

```
public boolean hasNonNoneActions()
```

- 统计非NONE动作的个数

```
public int hasNamNoneActions()
```

- 获取，设定资源用量

```
public ResourceUsage getResourceUsage()
public void setResourceUsage(ResourceUsage a_r) //获取这个PlayerAction 列表的资源用量
```

- ***添加/删除（单位，动作）***

```
public void addUnitAction(Unit u, UnitAction a)
public void removeUnitAction(Unit u, UnitAction a)
```

PlayerActionGenerator

用于列举玩家动作

- 构造函数，一个玩家所有可能的动作，以**choices列表**存储**<单位，该单位的可行动作>**

```
public PlayerActionGenerator(GameState a_gs, int pID)
```

- 打乱choices列表中的单位对应的可行动作

```
public void randomizeOrder()
```

- 返回随机动作

```
public PlayerAction getRandom()
```

PhysicalGameState

描述整个游戏的真实地图信息

- 构造函数，接收地图的长和宽(地形)

```
public PhysicalGameState(int a_width, int a_height)
```

```
PhysicalGameState(int a_width, int a_height, int t[])
```

- 长、宽、地形（terrain）的读取和设置（get，set）
- 添加/删除（remove）一个玩家，接收***Player***类型

```
public void addPlayer(Player p)
```

- 添加/删除一个单位，接收***Unit***类型

```
public void addUnit(Unit newUnit)
```

- 1、2获取地图***所有单位***、玩家。3、4获取指定玩家、单位（指定id）

```
public List<Unit> getUnits()
public List<Player> getPlayers()
public Player getPlayer(int pID)
public Unit getUnit(long ID)
```

- 获取一个方形区域的单位

```
public Collection<Unit> getUnitsAround(int x, int y, int squareRange)
public Collection<Unit> getUnitsAround(int x, int y, int width, int height) //x,y均为方形的左上角
```

- 获取单位（指定坐标）

```
public Unit getUnitAt(int x, int y)
```

- 判断游戏是否结束

```
boolean gameover()
```

- 返回赢家

```
public int winner() // -1:游戏未结束， unsigned int:玩家编号
```

- clone 函数

```
/**
    * Clone the physical game state, but does not clone the units The terrain
    * is shared amongst all instances, since it never changes
    */
public PhysicalGameState cloneKeepingUnits()

    /**
     * Clones the physical game state, including its terrain
     */
public PhysicalGameState cloneIncludingTerrain()

    /**
```

```
     * Clone the physical game state, but does not clone the units The terrain
     * is shared amongst all instances, since it never changes
     */
public PhysicalGameState cloneKeepingUnits()

    /**
     * Clones the physical game state, including its terrain
     */
public PhysicalGameState cloneIncludingTerrain()
```

- ***侦查地图中所有的空余点情况***

```
public boolean[][] getAllFree()
```

- 判断两PhysicalGameState对象是否类同

```
public boolean equivalents(PhysicalGameState pgs)
```

GameState

适用于`fully-observation`的游戏模式

- 构造函数，接收PhysicalGameState和UnitTypeTable，定义了游戏的**地图信息**和游戏可使用的**单位类型**

```
public GameState(PhysicalGameState a_pgs, UnitTypeTable a_utt)
```

- 获取当前的进行时间

```
public int getTime()
```

- 移除单位、获取玩家、获取单位、获取所有单位（与PhysicalGameState相同），尽量

```
public void removeUnit(Unit u)
public List<Player> getPlayers()
public Unit getUnit(long ID)
public List<Unit> getUnits()
```

- 获取单位的动作?，接收Unit类型

```
public UnitAction getUnitAction(Unit u)
```

- 获取`分配`给单位的动作?，接收Unit类型

```
public UnitActionAssignment getActionAssignment(Unit u)
```

- 判断是否玩家的所有单位有合法的动作

```
public boolean isComplete()
```

- 赢家、游戏结束（与PhysicalGameState相一致）

```
public int winner()
public boolean gameover()
```

- ***获取PhysicalGameState信息***

```
public PhysicalGameState getPhysicalGameState()
```

- ***判断一个坐标是否有被某单位占有，并且是否有单位将会移动到该坐标***

```
public boolean free(int x, int y)
```

- ***返回应用了上述条件的点的数组（对每个地图上的点判断）***

```
public boolean[][] getAllFree()
```

- 下达命令，可不由我们考虑，被回调

```
public boolean issue(PlayerAction pa)
public boolean issueSafe(PlayerAction pa)
```

- ***判断玩家是否有可执行动作***

```
public boolean canExecuteAnyAction(int pID)
```

- ***在UnitAction合法的情况下，检查动作之间是否冲突***

```
public boolean isUnitActionAllowed(Unit u, UnitAction ua)
```

- 获取玩家的动作列表

```
public List<PlayerAction> getPlayerActions(int playerID)
```

- 获取下一个动作完成的时间

```
public int getNextChangeTime()
```

- 运行一个游戏cycle，执行所有分配的动作，返回游戏是否结束

```
public boolean cycle()
public void forceExecuteAllActions()
```

- clone方法
- 获取所有动作的资源用量

```
public ResourceUsage getResourceUsage()
```

- *判断是否两个动作分配给了同一个单位*

```
public boolean integrityCheck()
```

示例

RandomAI

```
public PlayerAction getAction(int player, GameState gs) {
        try {
            if (!gs.canExecuteAnyAction(player)) return new PlayerAction();
            PlayerActionGenerator pag = new PlayerActionGenerator(gs, player);//内置的生成器
            return pag.getRandom();//使生成器返回一个随机操作
        }catch(Exception e) {
            return new PlayerAction();
        }
    }
```

RandomBiasedAI

```
public PlayerAction getAction(int player, GameState gs) {
        // attack, harvest and return have 5 times the probability of other actions
        PhysicalGameState pgs = gs.getPhysicalGameState();
        PlayerAction pa = new PlayerAction();

        if (!gs.canExecuteAnyAction(player)) return pa; //无路可走

        // Generate the reserved resources:
        for(Unit u:pgs.getUnits()) { //遍历场上每一个单位
            UnitActionAssignment uaa = gs.getActionAssignment(u); //获取单位的任务分配情况
            if (uaa!=null) {    //若已经分配过动作
                ResourceUsage ru = uaa.action.resourceUsage(u, pgs); //获取这个动作的资源使用量，资源使用量包含两部分(资源用量，所占坐标的集合)
```

```
13
 System.out.println(uaa.action.resourceUsage(u,pgs).toString());
14                  pa.getResourceUsage().merge(ru); //将动作的资源使用量添加入pa
15              }
16          }
17
18          for(Unit u:pgs.getUnits()) {
19              if (u.getPlayer()==player) {    //对己方操作
20                  if (gs.getActionAssignment(u)==null) {  //对没有进行分配的单
位操作
21                      List<UnitAction> l = u.getUnitActions(gs);  //获取单位在
gs下的动作列表
22                      UnitAction none = null;
23                      int nActions = l.size();    //统计动作数
24                      double []distribution = new double[nActions];   //初始
化动作的分布概率
25
26                      // Implement "bias":
27                      int i = 0;
28                      for(UnitAction a:l) {   //对指定动作赋予更高的选择概率
29                          if (a.getType()== UnitAction.TYPE_NONE) none = a;
30                          if (a.getType()== UnitAction.TYPE_ATTACK_LOCATION
||
31                                  a.getType()== UnitAction.TYPE_HARVEST ||
32                                  a.getType()== UnitAction.TYPE_RETURN) {
33                                  distribution[i]=BIASED_ACTION_WEIGHT;
34                          } else {
35                              distribution[i]=REGULAR_ACTION_WEIGHT;
36                          }
37                          i++;
38                      }
39                      try {
40                          UnitAction ua =
l.get(Sampler.weighted(distribution));  //依据概率选择动作
41                          //若选取的资源与当前的玩家所有已经分配的动作兼容，合并成已用
的资源
42                          if (ua.resourceUsage(u,
pgs).consistentWith(pa.getResourceUsage(), gs)) {
43                              ResourceUsage ru = ua.resourceUsage(u, pgs);
44                              pa.getResourceUsage().merge(ru);

45                              pa.addUnitAction(u, ua);    //有效动作，添加进pa
46                          } else {
47                              pa.addUnitAction(u, none);  //资源冲突，无效动
作，分配空操作
48                          }
49                      } catch (Exception ex) {
50                          ex.printStackTrace();
51                          pa.addUnitAction(u, none);
```

```
                    }
                }
            }
        }

        return pa;
    }
```

一个完整的简单案例

```
public class Test extends AbstractionLayerAI {
    UnitTypeTable m_utt = null;
    List<Unit> HarvestWorkers = new ArrayList<>();
    List<Unit> resources = new ArrayList<>();
    Unit my_base = null;
    Unit enermy_base = null;
    // This is the default constructor that microRTS will call:
    public Test(UnitTypeTable utt, PathFinding a_pf) {
        super(a_pf);
        m_utt = utt;
    }
    // This will be called by microRTS when it wants to create new instances of this bot (e.g., to play multiple games).
    public AI clone() {
        return new Test(m_utt, pf);
    }
    // This will be called once at the beginning of each new game:
    public void reset() {
    }
    // Called by microRTS at each game cycle.
    // Returns the action the bot wants to execute.
    public PlayerAction getAction(int player, GameState gs) {
        PlayerAction pa = new PlayerAction();
        PhysicalGameState pgs = gs.getPhysicalGameState();
        Player p = gs.getPlayer(player);
        //获取我方和敌方初始基地，二人游戏
        for (Unit u : pgs.getUnits()) {
            if (u.getType() == m_utt.getUnitType("Base") ){
                if (u.getPlayer() == player) {
                    my_base = u;
                }
                else if (u.getPlayer() != player){
                    enermy_base = u;
                }
            }
        }
        //获取资源数
        for (Unit u : pgs.getUnits()) {
```

```
            if (u.getType() == m_utt.getUnitType("Resource")) {
                resources.add(u);
            }
        }
        //挑选离我方基地近的资源作为开采点
        for (int j = 0; j <resources.size();++j){
            if (abs(resources.get(j).getX()-my_base.getX()) +
abs(resources.get(j).getY()-my_base.getY())
                    > abs(resources.get(j).getX()-enermy_base.getX()) +
abs(resources.get(j).getY()-enermy_base.getY())){
                resources.remove(j);
            }
        }
        //挑选后勤和先锋
        int harvestWorkerFind = 0;
        List<Unit> offendWorker = new ArrayList<>();
        for (Unit u : pgs.getUnits()) {
                if (u.getType() == m_utt.getUnitType("Worker") &&
                        u.getPlayer() == player) {
                    if (harvestWorkerFind < 2) {
                        HarvestWorkers.add(u);
                        harvestWorkerFind++;
                    }
                    else{
                        offendWorker.add(u);
                    }

                }
            }
        //生产矿工
        if (gs.getActionAssignment(my_base)==null && p.getResources() >
m_utt.getUnitType("Worker").cost){
            train(my_base, m_utt.getUnitType("Worker"));
        }
        //收获资源
        for (int j = 0; j < min(resources.size(),HarvestWorkers.size());
++j){
            if (gs.getActionAssignment(HarvestWorkers.get(j)) == null) {
                harvest(HarvestWorkers.get(j), resources.get(j),
my_base);
            }
        }
        //没有灵魂地进攻
        List<Unit> enermyUnits = new ArrayList<>();
        for (Unit u : pgs.getUnits()){
            if (u.getPlayer() != player && u.getPlayer() >= 0){
                enermyUnits.add(u);
            }
        }
```

```
        Integer s = new Random().nextInt(enermyUnits.size());
        for (Unit warrior: offendWorker){
            if (gs.getActionAssignment(warrior) == null &&
warrior.getType().canAttack){
                Unit target = enermyUnits.get(s);
                if (target == null){
                    break;
                }
                else{
                    attack(warrior, target);
                }
            }
        }
        //抽具化
        return translateActions(player, gs);
    }

    public List<ParameterSpecification> getParameters() {
        return new ArrayList<>();
    }
}
```