스타크래프트2

인공지능 봇 제작

팀 내가 제일 젤나가

제 1차 보고서

111600 박진원  
111599 김진호  
111626 서인국

Davaid Churchill 교수의 Command Center를 기반으로 제작하였습니다.

종족은 **프로토스**입니다.

1차 제출에는 기본적인 동작 위주,  
 2차 제출에는 고급 빌드와 컨트롤을 추가할 예정입니다.

<< 구성 >>

1. 빌드의 반복
2. ProductionManager에 renewBuildOrder() 메서드 추가

void ProductionManager::renewBuildOrder(const BuildOrder & buildOrder)

{

size\_t i(m\_queue.size());

for (i; i<buildOrder.size(); ++i)

{

m\_queue.queueAsLowestPriority(buildOrder[i], true);

}

}

* 기존의 setBuildOrder()의 경우, m\_queue를 초기화 하여서 빌드가 꼬이게 됩니다. 그렇기 때문에, m\_queue를 이어서 사용할 수 있는 renewBuildOrder() 메서드를 추가하였습니다.

1. 반복 생산문

* BotConfig.txt 파일에 LoopBuildOrder 태그를 추가하였습니다.
* 이에 맞추어서 StrategyManager.h에 Strategy 구조체에도 m\_loopOrder를 추가합니다.

struct Strategy

{

std::string m\_name;

sc2::Race m\_race;

int m\_wins;

int m\_losses;

BuildOrder m\_buildOrder;

BuildOrder m\_loopOrder;

Strategy();

Strategy(const std::string & name, const sc2::Race & race

, const BuildOrder & buildOrder, const BuildOrder & loopOrder);

};

* StrategyManager에 getLoopBookBuildOrder() 메서드 추가합니다.

const BuildOrder & StrategyManager::getLoopBookBuildOrder() const

{

auto buildOrderIt = m\_strategies.find(m\_bot.Config().StrategyName);

// look for the build order in the build order map

if (buildOrderIt != std::end(m\_strategies))

{

return (\*buildOrderIt).second.m\_loopOrder;

}

else

{

BOT\_ASSERT(false, "Strategy not found: %s, returning empty initial build order", m\_bot.Config().StrategyName.c\_str());

return m\_emptyBuildOrder;

}

}

* ProductionManager 의 manageBuildOrderQueue() 메서드 가장 윗줄에

// if there is nothing in the queue, oh well

if (m\_queue.isEmpty())

{

renewBuildOrder(m\_bot.Strategy().getLoopBookBuildOrder());

return;

}

을 추가합니다. 이로써 자동 생성 구문이 완성되었습니다.

1. 방어 스쿼드
2. CombatCommander의 updateDefenceSquadUnits()을 변경하였습니다.

void CombatCommander::updateDefenseSquadUnits(Squad & defenseSquad, const size\_t & flyingDefendersNeeded, const size\_t & groundDefendersNeeded)

{

auto & squadUnits = defenseSquad.getUnits();

// TODO: right now this will assign arbitrary defenders, change this so that we make sure they can attack air/ground

// if there's nothing left to defend, clear the squad

if (flyingDefendersNeeded == 0 && groundDefendersNeeded == 0)

{

defenseSquad.clear();

return;

}

size\_t defendersNeeded = flyingDefendersNeeded + groundDefendersNeeded;

size\_t defendersAdded = 0;

for (auto & unitTag : m\_combatUnits)

{

auto unit = m\_bot.GetUnit(unitTag);

BOT\_ASSERT(unit, "null combat unit");

if (!m\_squadData.canAssignUnitToSquad(unitTag, defenseSquad)

|| (m\_squadData.isAttackSquad(\*unit))

)

{

continue;

}

m\_squadData.assignUnitToSquad(unitTag, defenseSquad);

}

}

* 전투 유닛이 존재한다면 적의 유닛에 관계없이 방어를 시도하도록 변경하였습니다.
* 하지만, 만약 공격 스쿼드에 구성된 유닛이라면 방어를 위해 회군하지 않습니다.
* Squad 에 새로운 isAttackSquad() 메서드 추가

bool SquadData::isAttackSquad(const UnitTag & unit) const

{

const Squad \* unitSquad = getUnitSquad(unit);

return unitSquad->getName()=="MainAttack";

}

1. 감지 거리를 증가시켰습니다. (BaseLocation)

const int NearBaseLocationTileDistance = 40;

1. 일꾼 스쿼드
2. 일꾼의 방어 범위를 좁혔습니다.

* BaseLocation의 containsPosition()함수의 인자를 추가하였습니다.

// 0 - worker defence

// 1 -- combat defence

bool BaseLocation::containsPosition(const sc2::Point2D & pos, int situation) const

{

if (!m\_bot.Map().isValid(pos) || (pos.x == 0 && pos.y == 0))

{

return false;

}

if(situation==0)

return getGroundDistance(pos) < NearBaseLocationTileDistance / 4;

return getGroundDistance(pos) < NearBaseLocationTileDistance;

}

* updateScoutDefenceSquad()의 이름을 updateWorkerDefenseSquad()으로 변경
* 적 유닛의 숫자를 파악하기 위해 Squad의 size() 메서드를 추가하였습니다.

int Squad::size() const

{

return m\_units.size();

}

* updateWorkerDefenseSquad()의 최종버전

void CombatCommander::updateWorkerDefenseSquad()

{

// if the current squad has units in it then we can ignore this

Squad & workerDefenseSquad = m\_squadData.getSquad("ScoutDefense");

// get the region that our base is located in

const BaseLocation \* myBaseLocation = m\_bot.Bases().getPlayerStartingBaseLocation(Players::Self);

BOT\_ASSERT(myBaseLocation, "null self base location");

// get all of the enemy units in this region

std::vector<UnitTag> enemyWorkersInRegion;

for (auto & unit : m\_bot.UnitInfo().getUnits(Players::Enemy))

{

if (myBaseLocation->containsPosition(unit.pos, 0) && Util::IsWorker(\*m\_bot.GetUnit(unit))

)

{

enemyWorkersInRegion.push\_back(unit);

}

}

// get a workerDefenceSquad

if (workerDefenseSquad.size() <= enemyWorkersInRegion.size()) {

for (UnitTag enemyWorkerTag : enemyWorkersInRegion) {

// UnitTag enemyWorkerTag = enemyWorkersInRegion[i];

auto enemyWorkerUnit = m\_bot.GetUnit(enemyWorkerTag);

BOT\_ASSERT(enemyWorkerUnit, "null enemy worker unit");

UnitTag workerDefenderTag;

// get our worker unit that is mining that is closest to it

workerDefenderTag = findClosestWorkerTo(enemyWorkerUnit->pos);

if (enemyWorkerTag && workerDefenderTag)

{

// grab it from the worker manager and put it in the squad

if (m\_squadData.canAssignUnitToSquad(workerDefenderTag, workerDefenseSquad))

{

m\_bot.Workers().setCombatWorker(workerDefenderTag);

m\_squadData.assignUnitToSquad(workerDefenderTag, workerDefenseSquad);

}

}

}

}

// renew the workerDefenceSquad

if (!workerDefenseSquad.isEmpty() && enemyWorkersInRegion.size()==0)

{

for (auto & unitTag : workerDefenseSquad.getUnits())

{

auto unit = m\_bot.GetUnit(unitTag);

BOT\_ASSERT(unit, "null unit in scoutDefenseSquad");

Micro::SmartStop(unitTag, m\_bot);

if (Util::IsWorker(\*unit))

{

m\_bot.Workers().finishedWithWorker(unitTag);

}

}

workerDefenseSquad.clear();

}

}

1. 파일런 스페이싱

1) 첫 파일런을 미네랄과 넥서스 사이에 지어 다음 건물을 지을 수 없는 현상을 발견하여 이를 막기 위해 BuildingPlacer.cpp에

static int flag = 0;

선언 후 canBuildHereWithSpace에 다음을 추가 했습니다.

if (flag == 0 && b.type == 60)

{

for (int x = startx-2; x < endx+2; x++)

{

for (int y = starty-2; y < endy+2; y++)

{

if (!Util::IsRefineryType(b.type))

{

if (!buildable(b, x, y) || m\_reserveMap[x][y])

{

return false;

}

}

}

}

}

flag = 1;

return true;

1. 일꾼 포함 러쉬 (치즈 러쉬)
2. CombatCommander의 updateAttackSquadUnits()을 변경하였습니다.

if (!(unit->unit\_type == sc2::UNIT\_TYPEID::ZERG\_OVERLORD)

&& !(unit->unit\_type == sc2::UNIT\_TYPEID::ZERG\_QUEEN)

&& m\_squadData.canAssignUnitToSquad(unitTag, mainAttackSquad))

{

m\_squadData.assignUnitToSquad(unitTag, mainAttackSquad);

}

* !Util::IsWorker(\*unit) 부분을 삭제 하여 일꾼도 AttackSquad에 편성 되게 하였습니다.

1. Util의 IsCombatUnitType()을 변경하였습니다.

if (IsWorkerType(type)) { return true; }

* False 를 변경하여 Worker 유닛도 CombatUnit 취급을 하여 "CombatUnitsForAttack" 수의 포함 되게 하였습니다..

1. 기타
2. CombatCommander의 onFrame()의 실행시간을 미약하게나마 줄이기 위해 isSquadUpdateFrame() 함수 도입하였습니다.

bool CombatCommander::isSquadUpdateFrame()

{

if (m\_combatUnits.size() < 1) return false;

return true;

}