MicroRTS

Final Presentation

by

Ilias Baktybek, Akshay Patel, Matthew Berry, Miguel Quemado

Project Description

This project aims to create a bot capable of playing the real-time strategy game microRTS. We had initially developed a single bot together as both a means of establishing a baseline for the project as well a way to learn and understand the complex library. The bot is created through the microRTS library provided by the Farama Foundation, which includes the game in addition to the necessary classes, methods and algorithms. Eventually, we began developing our own individual bots that use different approaches and strategies. These different strategies offer insights into the game of microRTS and demonstrate varying elements of gameplay.

Tools

Source code management tool - Github

Build systems - Windows 11

Documentation tool - Doxygen, Javadoc

IDEs - Apache Netbeans

Static analysis tools - Apache Netbeans has a built-in feature for static analysis for Java

Unit-Testing tool - JUnit integration in Netbeans

Any other tools/libraries/SDKs/framework instrumental in your project - VS Code as code editor, Oracle JAVA, microRTS library by Farama Foundation

Contributions (Matthew)

I began by implementing the general structure for actions to be set based on unit type as well as initial code to limit action conflicts and began implementing the Barracks AI for the initial base AI. I then continued further developing a separate version of the AI where I experimented with several methods for improvement. The primary addition I made was creating a method for basing some decisions on simulations of potential future games. This allows varying levels of hard-coded processing (including completely simulated or completely hardcoded) to limit potential actions to a smaller set of actions with the final decision being made via simulations. I also created a process for more efficiently tracking state changes for easier prevention of conflicts. I continued implementing several improvements including refining the ranged unit, reworking worker action preprocessing, and implementing multithreaded simulations allowing more bot combinations to be simulated to avoid a perfection bias. Additionally, I prepared the demos including making the AIs work with the micro RTS GUI and setting up matches against the AIs.

Contributions (Akshay)

I started by working on our initial bot and added Time budgeting prototype functions and heavy unit functionality to the MINS AI bot. I then used the MINS bot as the base as we diverted into making our own AI bots and developed my AI bot to work as a passive defensive AI that has it's units work in tandem, as I hardcoded the heavy units to only go seek and attack when it has 2 light units with it, sort of like escorting the heavy unit and letting it tank damage while the light units attack the enemy units. I hard coded the resource gathering process for worker units and barack actions that will generate appropriate unit types for my strategy. I tried to prototype ranged units to work in my strategy but depending on the map sizes and game states they would bug out and either turn back towards my base or rush before the 1 heavy and 2 light unit setup was ready, removing the ranged units completely gave me more flexibility with the resources and made the unit generation faster for the other types. I also setup the slides for the checkpoint presentations.

Contributions (Ilias)

Throughout the semester, I have been developing a bot that combines the ideas of our initial bot and LightRush AI. The idea was to quickly deploy heavy units by building barracks quickly and not spawning too many workers. For the main attackers I chose heavy units. The strategy goes on by directing heavy units into the opponent's territory and attacking the nearest units they encounter. Along developing my bot I had issues with managing time budget and fixing bugs related to action conflicts. At this point, I managed to resolve those issues. Aside from developing my bot I had been doing some refactoring for our initial bot as well as implementing actions for light units at the beginning of the semester before we decided to develop our bots separately.

Contributions (Miguel)

I began by implementing some unit actions for the original version of the team MINS_Bot, specifically the ranged unit. As we began to develop our own bots, I decided to attempt to create a hard-coded implementation of a proxy light rush. This strategy aims to build a barracks ‘close’ to the enemy structures such as barracks and bases, then rapidly produce light units that seek out and attack the enemy. To implement this, I create a simple algorithm that decides where to place the ‘proxy’ barracks by judging location and size of the map. This algorithm has been tweaked and modified several times through intuition and testing. The biggest issue with this strategy is that it is entirely reliant on an initial strike, and is very susceptible to being stopped. Bots that mobilize quickly have the potential to destroy the base before it even has a chance to begin producing light units. I have been working on prototyping strategies to buy time for the barracks to produce lights, but have not seen much success. Ultimately, I don’t think this particular strategy is good for microRTS due to the fast-paced nature of the game. After multiple tests against multiple bots on varying maps, the bot’s success rate is low since it either fails to produce a barracks and combat units fast enough or the proximity of the barracks works against it and it gets eliminated quickly. I also cleaned up the repository a little.

Teamwork

-We had a member leave in the beginning who was responsible for writing some logic for light units, resource distribution and general worker actions, which meant we had to distribute work among the other group members and be ready for Checkpoint 2.

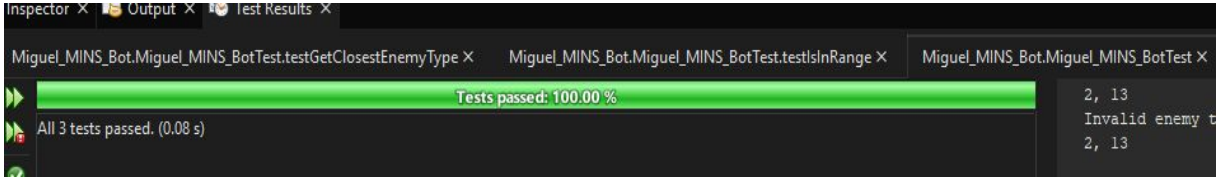
- Overall teamwork was good as we all helped each other learn and understand the microRTS library and helped each other solve problems we encountered

Test Cases

Matthew: I added several test cases using Junit (since it has built in compatibility with netbeans) for my version of the bots getAction() function to ensure the bot operated within time allotment and did not violate the integrity of the game state.

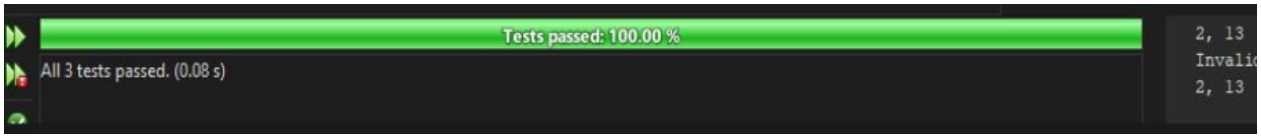
Ilias: I added few tests cases to ensure proper base actions and successful initialization stage as those play vital role in the overall gameplay. I used JUnit unit-testing framework because it properly suits our code written in Java.

Test Cases



Miguel: I used JUnit as it was integrated into the Netbeans IDE in a way that made it easy to work with. I made 3 test cases testing the functionality of posFree(), getClosestEnemyType() and isInRange(). For posFree it checked an open space and occupied space and should return true if open and false otherwise. I had 2 different scenarios in the test for this method. The inputs were an open space and an occupied space, both of which should return true and false respectively. getClosestEnemyType() should return the nearest enemy to a given Unit 'u'. If there are none, it should return itself, 'u'. Because of the way I had to initialize the game, there are no enemy units and as such returns the unit I had passed as input. isInRange() returns a boolean for if a unit is in range of another unit to attack/harvest/return to. I also had 2 scenarios for this, a unit in range of a base and a unit not in range of the base, both returning true and false respectively.

Test Cases



I used Junit for unit testing to test two functions, one function was for the base action that creates the worker units and the other is to have the worker units gather resources, as they are the basis of the entire resource economy and henceforth fighting units to be spawned. Initializes game state and gives the base and worker a task to perform.

```java
@Test
public void testWorkerActions() {
    // mock GameState
    PhysicalGameState pgs = new PhysicalGameState(8, 8); // assuming a 8x8 grid
    GameState gameState = new GameState(pgs, utt);
    Player player = new Player(0, 10); // Player ID 0 with 10 resources

    // Add a base and a worker for the player
    Unit base = new Unit(utt.getUnitType("Base"), player, 0, 0); // Base at position (0,0)
    Unit worker = new Unit(utt.getUnitType("Worker"), player, 1, 0); // Worker at position (1,0)
    pgs.addUnit(base);
    pgs.addUnit(worker);

    // Add a resource
    Unit resource = new Unit(utt.getUnitType("Resource"), null, 3, 3); // Resource at position (3,3)
    pgs.addUnit(resource);

    // Set up game state
    gameState.addPlayer(player);
    gameState.setPhysicalGameState(pgs);

    // Call the AI's getAction method
    PlayerAction playerAction = shayAI.getAction(0, gameState);

    // Check if the worker is performing a harvesting action
    boolean isHarvesting = false;
    for (Pair<Unit, AbstractAction> action : playerAction.getActions()) {
        if (action.m_a instanceof Harvest && action.m_b == worker) {
            isHarvesting = true;
            break;
        }
```

```java
@Test
public void testInitialization() {
    assertNotNull("Worker unit should not be null", shayAI.workerUnit);

}

@Test
public void testBaseActions() {
    // mock GameState
    PhysicalGameState pgs = new PhysicalGameState(8, 8); // assuming a 8x8 grid
    GameState gameState = new GameState(pgs, utt);
    Player player = new Player(0, 10); // Player ID 0 with 10 resources

    // Add a base for the player
    Unit base = new Unit(utt.getUnitType("Base"), player, 0, 0); // Assuming base at position (0,0)
    pgs.addUnit(base);

    // Set up game state
    gameState.addPlayer(player);
    gameState.setPhysicalGameState(pgs);

    // Call the AI's getAction method
    PlayerAction playerAction = shayAI.getAction(0, gameState);

    // Check if the action to train a worker has been issued
    boolean workerTrained = false;
    for (Pair<Unit, AbstractAction> action : playerAction.getActions()) {
        if (action.m_b.getType() == utt.getUnitType("Worker")) {
            workerTrained = true;
            break;
```

Demo

https://imgur.com/2CDVy3e

Left Game:
- Blue - Matthew's AI
- Red - Ilias's AI

Right Game:
- Blue - Shay's AI
- Red - Miguel's AI

Tournament Results:
https://github.com/OU-CS3560/microrts-f23/tree/main/Tournament%20Results

Additional Items for Grading

Meeting Attendance Sheet

11/21/23 - All members attended

11/28/23 - All members attended

12/03/23 - All members attended

12/05/23 - All members attended

Matthew Berry

Ohio ID, Github Username: mb135821, MatthewBerry135821

Contributions:

https://github.com/OU-CS3560/microrts-f23/commit/cea7400418ed53a941ac6f9152f2dc5206fefdd0

https://github.com/OU-CS3560/microrts-f23/commit/b0a2cdd283907c92b0132789f51ecdf52326d780

Akshay Patel

Ohio ID, Github Username: sp550519, Shayz614

Contributions: Bug fixed AI, added doxygen documentation html, Added Unit-testing file

- https://github.com/OU-CS3560/microrts-f23/commit/adbc4a90b0bffdafaf10a3b489c87d91ba424b38
- https://github.com/OU-CS3560/microrts-f23/commit/458eee8ef069db4d6cd6d54fb3a80a0fd98ca0b0
- https://github.com/OU-CS3560/microrts-f23/commit/84a47df69f26916b44f4a3ba0bb59786f9023acb
- https://github.com/OU-CS3560/microrts-f23/commit/f607e87288d92526dc0339178ea998c9156f0600

Ilias Baktybek

Ohio ID: ib873519, Github Username: iliasbaktybek

Contributions:
https://github.com/OU-CS3560/microrts-f23/commit/a5f26940d27a2cd10358643c3cb7d7f31969acf9

https://github.com/OU-CS3560/microrts-f23/commit/e1f1c52e151f5fa8a794e43b8827c1871e3fed3b

Miguel Quemado

Ohio ID: mq003322, Github Username: MQUEMADO16

Contributions:

https://github.com/OU-CS3560/microrts-f23/commits?author=MQUEMADO16

https://github.com/OU-CS3560/microrts-f23/commits/miguel-branch?author=MQUEMADO16