Continuous Integration

Group Name:
Cohort 1, Group 4

Group Number:
Cohort 1, Group 4

Members:
Jude Hall
Rosie Hogg
Ishraan Ismail
Sam Wildgust
Ruby Hanson
Tom Devany
Tomas Asllani

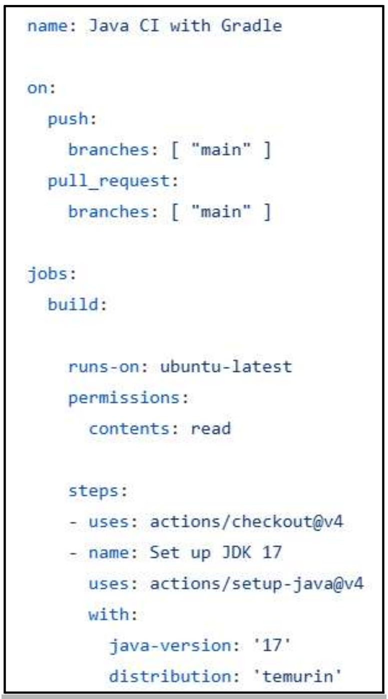
Continuous Integration Report

Continuous Integration Methods and Approaches

We used a number of methods to ensure the integration of the program was as smooth as possible, and that all team members knew what was going on and how to find the latest versions of the game at all times. These were:

- We made use of a github repository to store all code and files. Latest versions of the game were regularly uploaded to the repository. This meant that everything was stored in one place, and all team members knew how to find the latest version of the game.
- Team members made use of the actions tab in github to monitor whether builds had worked.
- We made use of automated builds, meaning that a single action could build the game and make it able to execute. This allowed us to save time and simplified the process of testing the game
- We used automated testing in place of manual testing in many instances, meaning issues with the game could be immediately found without having to find them manually, saving time and ensuring team members knew about any issues with the software
- We also kept a word document where each team member recorded what they had done immediately after programming. This ensured that members of the programming team knew exactly what other programmers had added to the game, and that other team members working on different parts of the project knew exactly what was in the game at all times

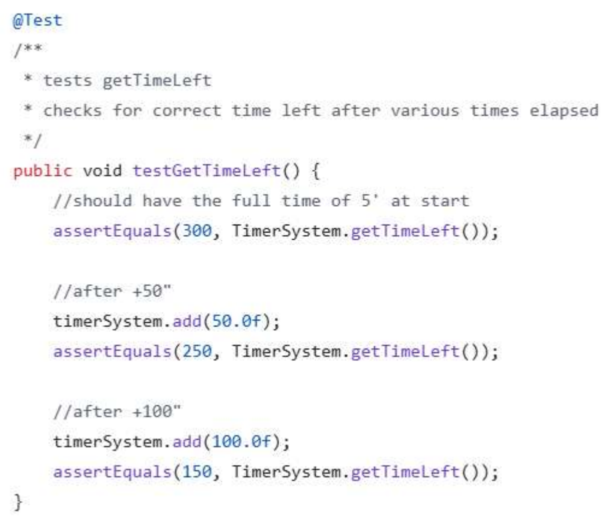
Continuous Integration Infrastructure



```yaml
name: Java CI with Gradle

on:
  push:
    branches: [ "main" ]
  pull_request:
    branches: [ "main" ]

jobs:
  build:

    runs-on: ubuntu-latest
    permissions:
      contents: read

    steps:
    - uses: actions/checkout@v4
    - name: Set up JDK 17
      uses: actions/setup-java@v4
      with:
        java-version: '17'
        distribution: 'temurin'
```

Part of the gradle.yml code for the automated build

We used the file github/workflows/gradle.yml to automatically build the game and ensure it builds correctly. First we added the high level structure of the workflow, using push and pull_request. The first job builds the program, compiling the java code. The second one analyses dependencies, getting a list of libraries that the code uses.



```java
@Test
/**
 * tests getTimeLeft
 * checks for correct time left after various times elapsed
 */
public void testGetTimeLeft() {
    //should have the full time of 5' at start
    assertEquals(300, TimerSystem.getTimeLeft());

    //after +50"
    timerSystem.add(50.0f);
    assertEquals(250, TimerSystem.getTimeLeft());

    //after +100"
    timerSystem.add(100.0f);
    assertEquals(150, TimerSystem.getTimeLeft());
}
```

Part of the TimerSystemTest.java file that contains the tests for the games timer

The majority of our testing was done through automated unit testing. The timer, movement, map, and score were all done through automated testing. This made it easy for team members to ensure these were working when building the code, and after they had changed the code. The example in the image is from the TimerSystemTest.java file, containing the tests for the game's timer. This screenshot in particular shows the automated test for checking if the amount of time left on the timer is correct It first checks that the time remaining is 300 seconds when the game begins, then checks that removing 50 seconds from the timer will work, and that removing 100 seconds will work, covering all instances of getting the time left used in the game.