Date: 13/01/25

UniTycoon
CONTINUOUS INTEGRATION REPORT

Group 6

TEAM6 Game Studios

Work Undertaken by:

Thomas Koukouris
Adam Khan
Oliver Herron
Sam Jordan
Nathan Hopper
Fergus Irvine

Continuous Integration Report

i. Continuous Integration Approach

In this section, we planned the approach used for incorporating Continuous Integration into the development process of our game ‘UniversityTycoon’. This was achieved in order to maintain consistency without lowering quality and to prevent more manual errors while keeping a flowing workflow with the entire team. This approach was important given the iterative nature of the project, where new features and fixes were added frequently, necessitating constant validation of the codebase.

The continuous integration pipeline was implemented using GitHub Actions and cloud-based functionality to automate builds and testing processes. We selected this platform because of the simple way it integrates with our GitHub repository. This implies that the code would have a fixed state following each commit, with the early identification and alerting of such error events from automated techniques.

Key Highlights of the Continuous Integration approach:

- GitHub as the version control system and repository provider
- Gradle as the primary build tool
- JUnit for automated testing
- Continuous Integration pipeline activated on push and pull requests to the main branch
- Comprehensive reporting and logging mechanisms

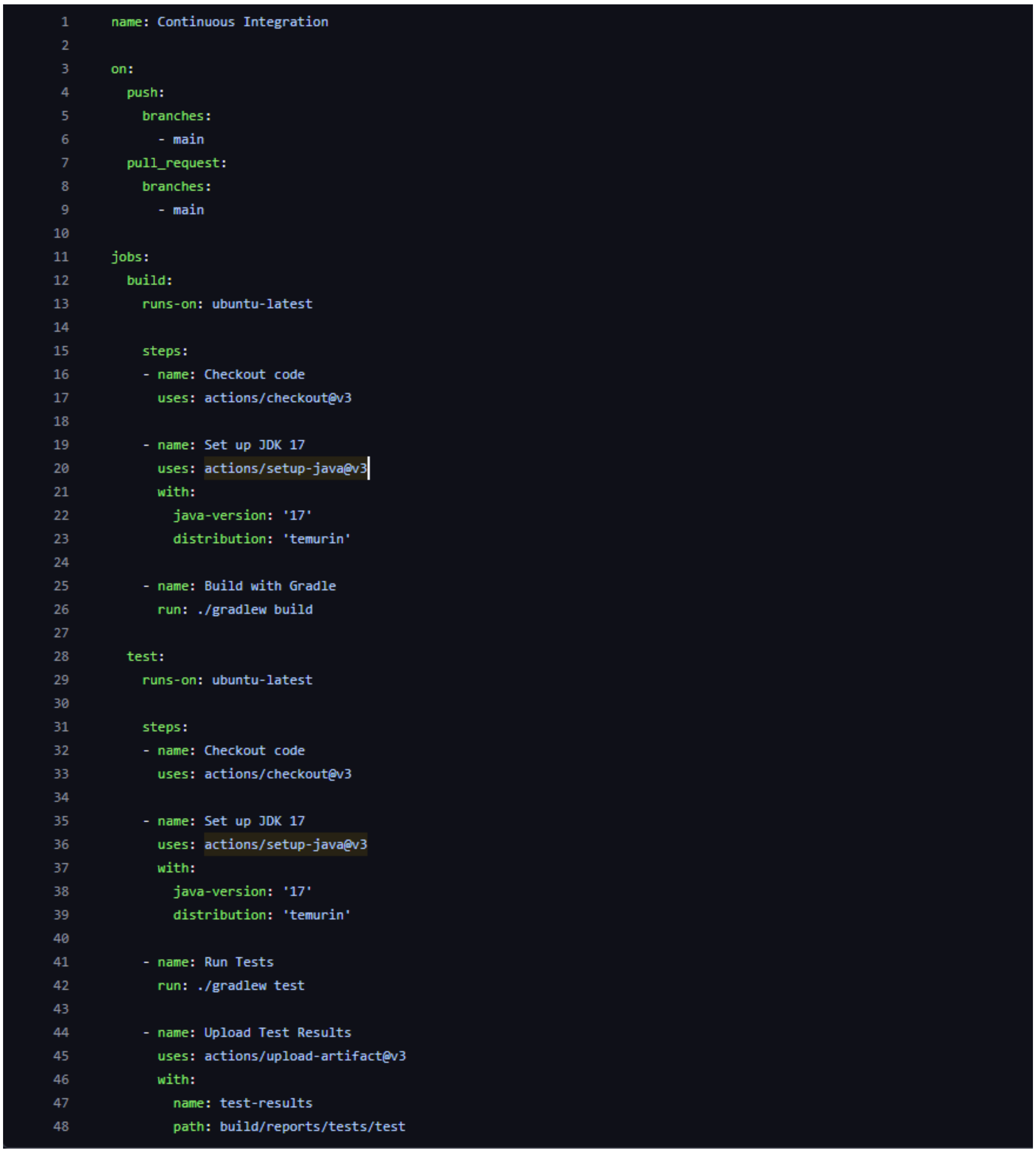
ii. Continuous Integration Infrastructure

Pipeline Overview The CI pipeline was optimized to be both powerful and easy to use, automating key elements and delivering fast feedback to the team. Following is a description of the steps used by the pipeline:

1. **Build Stage**
    - ***Tool***: Gradle
    - ***Action***: Compiles the project and resolves dependencies
    - ***Explanation***: Gradle was chosen primarily for its dependency handling and for its integration into the libGDX framework.
    - Outcome: Ensures the integrity of codebase by checking whether there is a build error.
2. **Test Stage**
    - ***Tool***: JUnit
    - ***Action***: Runs unit tests to verify proper function of key game elements.
    - ***Explanation***: JUnit offered a powerful testing environment for game logic, e.g., house layout, events, satisfaction value calculations, etc. Automating tests guaranteed no unintended breakage of the existing functionality.
3. **Code Review Stage**
    - ***Tool***: Visual Studio Code
    - ***Action***: Developers manually reviewed the code for syntax errors and logic flaws

- ***Explanation***: Given the team's reliance on an IDE for development, manual reviews in VSCode allowed for a pragmatic approach to identifying errors quickly.
- ***Outcome***: Checks to catch errors which are obvious before the code is committed to the repository.

iii. Workflow Configuration The pipeline configuration was defined using a GitHub Actions YAML file. The configuration is below:



```yaml
name: Continuous Integration

on:
  push:
    branches:
      - main
  pull_request:
    branches:
      - main

jobs:
  build:
    runs-on: ubuntu-latest

    steps:
    - name: Checkout code
      uses: actions/checkout@v3

    - name: Set up JDK 17
      uses: actions/setup-java@v3
      with:
        java-version: '17'
        distribution: 'temurin'

    - name: Build with Gradle
      run: ./gradlew build

  test:
    runs-on: ubuntu-latest

    steps:
    - name: Checkout code
      uses: actions/checkout@v3

    - name: Set up JDK 17
      uses: actions/setup-java@v3
      with:
        java-version: '17'
        distribution: 'temurin'

    - name: Run Tests
      run: ./gradlew test

    - name: Upload Test Results
      uses: actions/upload-artifact@v3
      with:
        name: test-results
        path: build/reports/tests/test
```