**Team Project Sprint #3**

Instructions

Please read the instructions carefully. All members of your team should discuss the instructions together to ensure that everyone is on the same page.

**Objectives**

1. Update and complete the user stories and acceptance criteria of the target software that allows a human player to play against either a human or a computer opponent.
2. Complete the implementation of all user stories, including improvements on the previous sprint. Note that the computer opponent should make a reasonable attempt at beating the human player. For example, it can defeat the worst human player.
3. Review the software design of the final production code.

* Summarize the user-interface design, using a combination of screenshots and textual descriptions.
* Describe the software architecture of the final production code using class diagram(s). It should capture the main classes and their relationships.
* Describe the algorithms for the computer opponent to place a piece, remove a piece, move and fly a piece. The descriptions should be understandable without referring to the source code. For example, you may use pseudo code.
* Discuss how your code can be extended for rule variations and other similar board games. What classes and methods need to be changed, and how? How was the Open-Closed Principle applied (i.e., which functions or classes are open for extension, but closed for modification)?

1. Conduct a code review exercise as a team for at least one hour and report the findings. Everyone should have read the following instructions before the exercise.

In this exercise, the team will apply code review practices to one or two most important classes of the game (and other classes if time permits) in the team project. One member should lead the code review and another member should take notes and document the findings. The review should be constructive. Do not criticize the developers who wrote the code under review. There is no need to test the program or make changes during the review exercise. The findings can be used to improve your project after the review.

In addition to looking for bugs, the review should check: (1) whether the entire project has followed the coding standard in a consistent manner, (2) whether the project has followed the design principles introduced in class, and (3) whether there are code smells that indicate the need for refactoring. The following checklists provide basic guidelines. You may add new items to each of the checklists.

Checklist #1: Coding Standards

* Are there any violations of naming conventions?
  + Packages, classes, methods, variables, constants
  + Production code vs test code
* Is the ordering convention of method arguments followed in each method?
* Are all the comments meaningful and valid?
  + Are the precondition and post-condition of each method documented?
* Is the same style used for all curly braces of code blocks?
* Consistent indentation?
* …

Checklist #2: Design Principles

* Does each class have a good abstraction and good class interface?
* Is the visibility of each variable, method, and class (private, protected, public, default) appropriate?
* Is the command-query separation principle violated?
* Design by contract: for each public method, is Design by Contract followed? If so, is the specified precondition reasonable and available?
* Is the Open-Closed Principle violated?
* Is the Single-Responsibility Principle violated?
* …

Checklist #3: Code Smells

* Is there any magic number or unnamed constant?
* Is there any unnecessary global or class variable?
* Is there duplicate code?
* Are there long methods?
* Does any method have a long parameter list?
* Is there any over-complex expression?
* Is there any switch or if-then-else statement that should be replaced with polymorphism?
* Is there any variable or method name whose intent is unclear?
* Are there similar methods in multiple classes?
* …

Checklist #4: Secure Coding

* Strong password policy
* Prevention of SQL injection
* Integer overflow
* …

1. Summarize lessons learned. Everyone needs to submit a separate summary of lessons learned from this team project, answering the following questions:

* What did you personally gain from the project?
* What does your project do well, and what could your project do better?
* How could you improve your development process if you develop a similar game from scratch?

**Deliverables and Grading Policy**

1. Project Report (**35 points**)

The project report should include the following sections:

* 1. Updated complete user stories using the template discussed in class. **(1 points)**

Provide a complete list of user stories and estimated/actual efforts for the target software that allows a human player to play against either a human or a computer opponent.

* 1. Updated complete acceptance criteria using the template discussed in class. **(2 points)**

Provide complete acceptance criteria for all the user stories.

* 1. Implementation tasks **(12 points)**

Describe the production code, automated test code or manual test cases for all the user stories. For each acceptance criterion of every user story, you need to implement at least one test (either test code or manual test case).

* 1. The design of the final production code (**8 points**)

User-interface design (**2 points**)

Software architecture (**2 points**)

Algorithm (**2 points**)

Extensibility (**2 points**)

* 1. Findings of the code review exercise **(2 points)**
  2. Minutes of ALL meetings, including, but not limited to: project/sprint planning meeting, stand-up meeting, backlog grooming, retrospective meeting, and pair programming (or development) session. **(2 points)**
  3. A table of buddy ratings. Individual members may email their buddy ratings to the instructor or teaching assistant.
  4. Lessons learned (individual submission, **2 points**)

Each team only needs to submit one report. For an individual member to receive the credit for this part of the project, the team’s project report must include explicit evidence of his/her contribution (e.g., his/her name is listed as a developer).

1. Demonstration (**10 points**)

Within 15 minutes, clearly demonstrate that:

1. your project has completed the implementation of all required features.
2. for each acceptance criterion of an implemented user story, your project has implemented either an automated test method or performed an acceptance test manually.
3. your project has some unique features or enhancements (optional).

Grading of the demonstration: completion of the required functions (**5 points**), and overall presentation (**5 points**) using the following evaluation rubric:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Poor | Fair | Good | Very Good | Excellent |
| Was the demonstration logically organized |  |  |  |  |  |
| Were points made clearly and concisely |  |  |  |  |  |
| Were the grader or instructor’s questions answered satisfactorily |  |  |  |  |  |

3. Source Code

Submit all source code. Make sure your project report is consistent with the source code.

**Team Project Sprint #3**

Report Template

Team Name:

Team Members:

1. **Updated User Stories**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **ID** | **User Story Name** | **User Story Description** | **Priority** | **Estimated effort (hours)** | **Actual effort (if completed)** | **Status (completed, toDo, inProgress)** | **Developer names** |
| 1 |  |  |  |  |  |  |  |
| 2 |  |  |  |  |  |  |  |
| .. |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |

1. **Updated Acceptance Criteria (AC)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **User Story ID and Name** | **AC**  **ID** | **Description of Acceptance Criterion** | **Status (completed, toDo, inPprogress)** | **Developer Names** |
| 1 story one | 1.1 |  |  |  |
|  | 1.2 |  |  |  |
|  | … |  |  |  |
| 2 story two | 2.1 |  |  |  |
|  | … |  |  |  |

1. **Updated Implementation Tasks**

Include the tasks from the previous report and highlight the new tasks with a different color.

Summary of production code

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **User Story ID and Name** | **AC ID** | **Class Name(s)** | **Method Name(s)** | **Developer Name(s)** | **Status** | **Notes (optional)** |
| 1 | 1.1 |  |  |  |  |  |
|  | 1.2 |  |  |  |  |  |
|  | … |  |  |  |  |  |
| 2 | 2.1 |  |  |  |  |  |
|  | … |  |  |  |  |  |

Summary of automated test code (directly corresponding to some acceptance criteria)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **User Story ID and Name** | **Acceptance Criterion ID** | **Class Name (s) of the Test Code** | **Method Name(s) of the Test Code** | **Description of the Test Case (input & expected output)** | **Status** | **Developer Name(s)** |
| 1 | 1.1 |  |  |  |  |  |
|  | 1.2 |  |  |  |  |  |
|  | … |  |  |  |  |  |
| 2 | 2.1 |  |  |  |  |  |
|  | … |  |  |  |  |  |

Summary of manual test cases (directly corresponding to some acceptance criteria)\

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **User Story ID and Name** | **Acceptance Criterion ID** | **Test Case Input** | **Test Oracle (Expected Output)** | **Status** | **Notes** | **Developer Name(s)** |
| 1 | 1.1 |  |  |  |  |  |
|  | 1.2 |  |  |  |  |  |
|  | … |  |  |  |  |  |
| 2 | 2.1 |  |  |  |  |  |
|  | … |  |  |  |  |  |

Summary of other automated or manual tests (not corresponding to the acceptance criteria)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Number** | **Test Input** | **Expected Result** | **Class Name of the Test Code** | **Method Name of the Test Code** | **Status** | **Developer Name(s)** |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

1. **Design Documentation**
2. **User Interface Design**

List the names of the team members who contributed to this section.

Summarize the user-interface design, using a combination of screenshots and textual descriptions.

1. **Software Architecture**

List the names of the team members who contributed to this section.

Provide a class diagram that captures the main classes and their relationships in your final program.

1. **Algorithm Design**

List the names of the team members who contributed to this section.

Describe the algorithm design of the computer opponent (e.g., using pseudo code). The description should be understandable without referring to the source code.

1. **Extensibility**

List the names of the team members who contributed to this section.

Discuss how your code can be extended for the variants of Nine Men's Morris, including Six Men’s Morris and Twelve Men’s Morris. What classes and methods need to be changed, and how? How was the Open-Closed Principle applied (i.e., which functions or classes are open for extension, but closed for modification)?

1. **Findings from the Code Review Exercise**

Use the following template to document the findings from the code review of **each** class.

Participant names:

Class that was reviewed:

|  |  |  |  |
| --- | --- | --- | --- |
| **Checklist** | **Checklist Item** | **Findings** | |
| Coding Standards | Naming conventions |  | |
| Ordering convention of method arguments |  | |
| Meaningful and valid comments |  | |
| Consistent style of code blocks |  | |
| Consistent indentation |  | |
| … |  | |
| Design Principles | Good class abstraction and interface |  | |
| Appropriate visibility of each variable, method, and class |  | |
| Any violation of the command-query separation principle |  | |
| Design by contract (pre/post-conditions) |  | |
| Is the Open-Closed Principle violated? |  | |
| Is the Single Responsibility Principle violated? |  | |
| Code Smells | Magic numbers |  | |
| Unnecessary global / class variable |  | |
| Duplicate code |  | |
| Long methods |  | |
| Long parameter list |  | |
| Over-complex expression |  | |
| Switch or if-then-else that needs to be replaced with polymorphism |  | |
| Variable or method name whose intent is unclear |  | |
| Any similar methods in other classes? |  | |
| … |  | |
| Secure Coding | Strong password policy |  | |
| Prevention of SQL injection |  | |
| Integer overflow |  | |
| … |  | |
| **Bugs** | **Buggy code snippet** | **What is the bug?** | **Why it is a bug?** |
|  |  |  |
|  |  |  |
|  |  |  |

1. **Meeting Minutes (only during this sprint)**

Report the minutes of all meetings, including, but not limited to: project/sprint planning meeting, stand-up meeting, backlog grooming, retrospective meeting, and pair programming session.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Date** | **Time and Duration** | **Place** | **Participant Names** | **Purpose of the Meeting** | **Specific Action Items** |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

1. **Buddy Ratings**

If you don’t feel comfortable to include your ratings in this report, you may email your ratings to the instructor or grader.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *Rating giver* | *Rating receiver* | | | | |
|  | Full Name 1 | Full Name 2 | Full Name 3 | Full Name 4 |
| Full Name 1 | X |  |  |  |
| Full Name 2 |  | X |  |  |
| Full Name 3 |  |  | X |  |
| Full Name 4 |  |  |  | X |
|  | *Average* |  |  |  |  |