Project Overview

You will create the ***Quoridor*** application for the well-known board game Quoridor in two major phases. The detailed rules of the game can be found at https://en.wikipedia.org/wiki/Quoridor and at https://quoridorstrats.wordpress.com/beginners-guide-rules-and-basics/.

Phase 1

When the *starting of a new game* is initiated, the two players playing white and black *provide a new (unique) user name* or *select from existing user names* (used in previous games) and we *set the total thinking time* for each player. Then the Quoridor *board is initialized* with 10 walls at the stock of each player and the clock starts for White. In this first phase, the players can only place walls by *grabbing a wall* from his/her stock, *rotating the wall* and *moving the wall* to the designated place and then to finalize wall placement (by *dropping the wall*). The game shall be checking that a wall is not overlapping with other walls before accepting the final placement of a wall. For testing purposes, the inverse functionality of picking up a wall is also worth being implemented.

The game shall also support to *load a position* and *save a position* as a textual file using an algebraic notation. The game shall be able to *check if an invalid position* is attempted to be loaded where walls overlap with each other and prevent the load in such a case.

In this algebraic notation, each square gets a unique letter-number designation. Columns are labeled ***a*** through ***i*** from the white player's left and rows are numbered 1 through 9 from Black player's side to White's side. Initially, White starts from ***e9*** and Black starts from ***e1***. Each wall is defined by the square directly to the northwest of the wall center from White player's perspective, as well as an orientation designation. For example, a ***vertical*** wall between columns ***e*** and ***f*** and spanning rows ***3*** and ***4*** would be given the designation ***e3v***.

A text file representing a position has exactly two lines (see below): the first line starts with a ***W:*** prefix if white is next to move (and starts with a ***B:*** prefix otherwise) contains a comma-separated list starting with the position of the White player followed by the list of walls the White player has placed on the board. The second line starts with the other player's prefix, (e.g. ***B:***) followed by the other player's position and the list of walls placed by him/her. For example, if it is White to move, the following file describes a valid game position.

```
W: e9, e3v, d3v
B: e1, b6h
```

If it is Black to move, the following file describes a valid game position.

```
B: e1, b6h
W: e9, e3v, d3v
```

Phase 2

In this second phase of the project, the game shall enable to *move pawn* to an adjacent square as well as to enable *jump pawn* over the opponent where the correct behavior is specified using statemachines. As a complex feature, the game should *check if a path exists to the target area* for each player in the current position. This feature is a corner case of correct wall placement (partially) addressed in Phase 1.

The game shall also *identify if a player won* by reaching the designated target area (i.e. the first row on the opposite side of the board). The game shall enable for a player to *resign the game* or *offer a draw*, which can be accepted or refused by the opponent. After that, the game shall *report the final result*, and the current game is no longer running.

Furthermore, additional features should include to *load a game* and to *save a game* in an algebraic notation. Each pawn move is defined by the new square occupied by the pawn. Each fence move is defined by the square directly to the northwest of the wall center from White player's perspective, as well as an orientation designation. Games are notated as they are in chess: by the move number, player 1's move, and player 2's move. For example,

```
1. e8 e2
2. e7 e3
3. e3h e6h
```

As a related feature, the game shall support to *replay a game* (in replay mode) by supporting functionality such as *step forward*, or *step backward* one move, or *jump to start position*, or *jump to end position*.

1 Deliverables

You will deliver the ***Quoridor*** application in several iterations throughout the term. For each iteration, one or more deliverables are to be submitted as described below. More detailed information will be provided for each iteration. The project is done in a ***team of six students***. If the number of students in the class is not divisible by six, then some teams may consist of five or seven students with the instructor's permission. This permission will only be granted once the final number of students is known after the add/drop deadline on **Tuesday, September 17, 2019**. Please take into account that groups of five or seven students are exceptions and not the norm.

Generally, the whole team is responsible for each deliverable. For Iterations 2-5, individual team members are responsible for some specific deliverables.

Deliverables for Iteration 1 – Domain Model (4%) (due Sunday, September 29, 2019 23:30)

- Use Umple to define the domain model showing all concepts and relationships of the Quoridor

game including features to be developed in Phase 1 and Phase 2.
- Generate code from the domain model and commit the code to your group's repository in the GitHub organization of the course (https://github.com/McGill-ECSE223-Fall2019/)

Deliverables for Iteration 2 – UI Mockup, Controller Interfaces, Mapping of Gherkin Scenarios (6%) (due Friday, October 11, 2019 23:30)

- For each feature of the game (i.e. all features of Phase 1 and Phase 2), you will need to create a mockup of the user interface and document it in the wiki of your Github repository.
- As an input for this deliverable, you will receive scenarios written in the Gherkin language, which capture the main interactions during the game. Core features of Phase 1 will be the following: **start a new game**, **provide or select user name**, **set total thinking time, initialize board**, **rotate wall**, **grab wall**, **move wall**, **drop wall**, **save position**, **load position**, **validate position**, **switch player (aka. update board)**
- Assign the development of the features to your team members, i.e., each team member is individually responsible for two specific features.
- For each feature individually specify the Controller interface as needed to realize the feature.
- For each feature, individually map the corresponding Gherkin scenarios to acceptance tests to be executed by Cucumber in the context of your Controller interface, and the functions of the user interface. Obviously, these tests will fail at this stage, but as you start implementing features, more and more tests will gradually pass.

Deliverables for Iteration 3 – Implementation (10%) (due Sunday, November 3, 2019 23:30)

- Implement the assigned features individually and as a team in Java as described in Iteration 2 and commit the code to your group's repository in the course's GitHub organization
- At this stage, your application should already be partially playable
- All related acceptance tests should successfully execute.

Deliverables for Iteration 4 – State Machine Specification and Implementation, Updated Controller Interface, Mapping Gherkin Scenarios (8%) (due Sunday, November 17, 2019 23:30)

- This iteration provides Gherkin specification of three features developed as a team: **move pawn**, **jump pawn**
- As a team, use Umple to define the state machine to **move pawn** and to **jump pawn**.
- Generate code from the state machine(s) and commit the code to your group's repository in the course's GitHub organization.
- Implement the Controller part of the assigned methods of the state machine (related to events, actions, guards) individually and as a team in Java and commit the code to your group's repository in the course's GitHub organization
- Moreover, you will receive Gherkin specification of further features to be developed individually: **check if path exists**, **identify if game won**, **identify if game drawn**, **report final result**, **resign game**, **load game**, **save game**, **enter replay mode, step forward**, **step backward**, **jump to start position**, **jump to final position**.
- Assign the development of these features to your team members, i.e., each team member is individually responsible for one specific feature. For each feature, specify the Controller interface as needed to realize the feature.
- For each feature, individually map the corresponding Gherkin scenarios to acceptance tests to be executed by Cucumber in the context of your Controller interface, and the functions of the user interface. Obviously, these tests will fail at this stage, but as you start implementing features, more and more tests will gradually pass.

Deliverables for Iteration 5 – Final Application (8%) (due Sunday, December 1, 2019 23:30)

- Complete the implementation of the assigned features individually and as a team in Java as described in Iteration 4 and commit the code to your group's repository in the course's GitHub organization.

- You should ensure that all related acceptance tests should successfully execute.
- As a team, correct any mistakes in the implementation from earlier iterations

Deliverables for Iteration 6 – Demo (4%) (due Monday/Tuesday, December 2/3, 2019)

- Give a demo of your application to the instructor.
- The upload of presentation slides of the demo is due on **Sunday, December 1, 2019, 23:30**.
- Time slots will be available by Monday, November 25th, 2019 on a first-come-first-served basis.

2 User Stories for Key Features of the Game

As an initial input, user stories are provided for the key features of the game. However, we strongly recommend that you used other public sources to learn about the Quoridor game.

Phase 1:

Start a new game: As a Quoridor player, I want to start a new game of Quoridor against some opponent.
Provide or select user name: As a player, I wish to use my unique user name when a new game starts, or create a new user name if I haven't played the game before.
Set total thinking time: As a player, I wish to set the total thinking time (minutes and seconds) for each player to ensure that a game does not last forever.
Initialize board: As a player, I want to see the Quoridor board in its initial position and my stock of walls and my clock is counting down so that I can start playing the game.
Grab wall: As a player, I want to grab one of my walls in my stock to initiate wall placement as a move.
Move wall: As a player who grabbed a wall, I wish to move the wall between possible rows and columns of the Quoridor board so that I could move it to its designated target position. I wish to get feedback from the game if a designated wall position is illegal.
Rotate wall: As a player who grabbed a wall, I want to rotate the wall by 90 degrees (from horizontal to vertical or vice versa) to adjust its designated target position. I wish to get feedback from the game if a designated wall position is illegal.
Drop wall: As a player, I wish to drop a wall after I navigated it to a designated (valid) target position in order to register my wall placement as my move.
Save position: As a player, I want to be able to save the actual position during a Quoridor game so that I can continue the game at a later stage from the exact position.
Load position: As a player, I want to continue a game from a given position as if this position were the current (starting) position of a new game.
Validate position: As a player, I want to avoid loading invalid positions with e.g. overlapping walls or out-of-track pawn or wall positions.
Switch player (aka. Update board): As a player, I wish to know when it is my turn to move, and I want to see an updated board once my opponent finished his last move. My clock shall be counting down then. Once my move is finished, my clock shall be stopped and it shall be my opponent's turn.

Phase 2:

Move pawn: As a player, I want to move my pawn to an adjacent square assuming that my pawn still stays on the board after the move, and this adjacent square is not blocked by existing walls.
Jump pawn: As a player, I want to jump my pawn over my opponent's pawn if there are now walls blocking the jump in that direction (horizontal walls for up/down jump, vertical walls for left/right jump).
Check if path exists (to target area): As a player, I want the app to verify current block placement that there at least one path is still available that leads to my target area (i.e. the first row of my opponent).
Identify if game won: As a player, I want the Quoridor app to identify when a game is won by one of the players (I reached target area or opponent's time is up), and the game should be stopped immediately.

Identify if game drawn: As a player, I want the Quoridor app to identify if a game has a threefold repetition of moves (when the game is drawn) and stop the game immediately.
Report final result: As a player, I want to be notified when I won, lost or draw a Quoridor game.
Resign game: As a player, I wish to resign a game if I am sure that my opponent has a winning position.
Load game: As a player, I want to load a previously played game recorded in an algebraic notation in a text file so that I can continue or review it. If the game is not yet finished, I wish to continue playing from the final position.
Save game: As a player, I want to save the current game in a text file even if the game has not yet been finished so that I can continue or review it later.
Enter replay mode: As a player, I wish to review a past game in replay mode to walk through the moves.
Step forward: As a player using replay mode, I want to check the next move made by the next player in turn and see the board position after that move.
Step backward: As a player using replay mode, I also want to check the position before the last move by taking (half) a step backwards.
Jump to start position: As a player using replay mode, I wish to scroll fast to the very beginning of the game.
Jump to final position: As a player using replay mode, I wish to scroll fast to the very end of the game.

3 Bonus Features

The following list of features of the Quoridor application can be considered as a bonus feature. The corresponding deadline of the respective features are also given.

- **B1: Four-player mode (Deliverables 3 and 5 – Del 3 and Del 5)**: This feature supports to play Quoridor with four persons instead of two. This does not involve the use of any new technologies, so it is an easy way for bonuses.
  Algorithmic difficulty (AD): 2, Technological difficulty (TD): 1
- **B2: Hints (Del 5):** This feature provides the user various hints during gameplay upon request (e.g. whether to move or place a wall, which way to move, how far he/she is from target area, etc.).
  AD: 3, TD: 2
- **B3: Network/LAN-mode (Del 5):** This feature enables to play a Quoridor game where one of the players is using a remote computer, thus network communication is necessitated. This feature can be demonstrated by running the same Quoridor application on different computers (i.e. the program developed by one team), or to play across Quoridor applications developed by two different teams, which involves extra integration effort.
  AD: 2, TD: 4
- **B4: Computer gameplay (Del 5)**: This feature allows to play a two-player Quoridor game against the computer. The teams can implement any AI-based approaches for computer play (ranging from best-first search or minimax algorithm to combined ML techniques like AlphaQuoridor 😊).
  AD: 5, TD: 2-4 (depending on what technologies are used).
- **B5: Use Eclipse Modeling Framework (EMF) as domain model (Deliverable 3 / Del3)**: This feature would use an alternate code generation technology (called Eclipse Modeling Framework, EMF) instead of Umple to derive the domain model used by your application. EMF is an open source framework, which may already be deployed to your Eclipse environment, and it is available at: https://www.eclipse.org/modeling/emf/.
  AD: 3, TD: 5
- **B6: Use Yakindu Statechart Tools for statemachine implementation (Del 4)**: This feature uses an industrial statechart modeling and code generation technology (Yakindu Statecharts) instead of Umple to provide the executable behavior for the required features. We have received academic

licenses for Yakindu Statecharts, so please contact the instructor if you wish to address this feature. **AD: 3, TD: 5.**

Note that some of these features requires significant extra implementation efforts from your team. Moreover, bonus features do not pay well compared to compulsory features, so there is no point in implementing 5 bonus features if your team loses many marks on compulsory tasks.

Griffin groups are required to complete B6 and one more bonus feature selected by them.

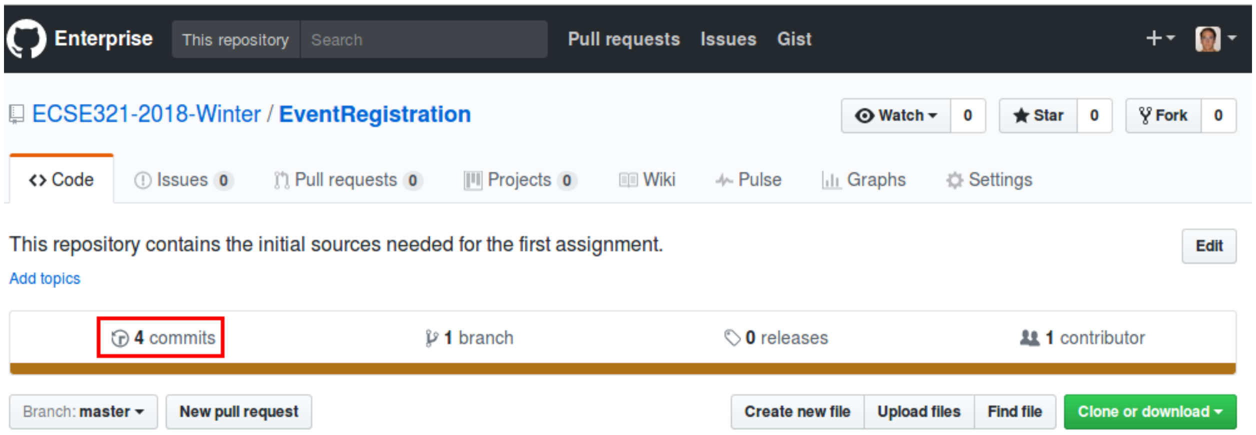
4 Technology Constraints

Your ***Quoridor*** application must be implemented in Java with a suitable framework for the user interface. The UI does not have to be Java. However, the use of technologies other than Java Swing and Java 2D for the UI must be approved by the instructor. Your domain model and state machines must be specified with Umple and code generated from them with Umple. The use of other state machine modeling and related code generator technologies needs approval by the instructor. In all cases, you must use the generated code in your application. You are not allowed to make manual modifications to the generated code, but you may add native Java code to the domain model specified with Umple.

5 General Rules

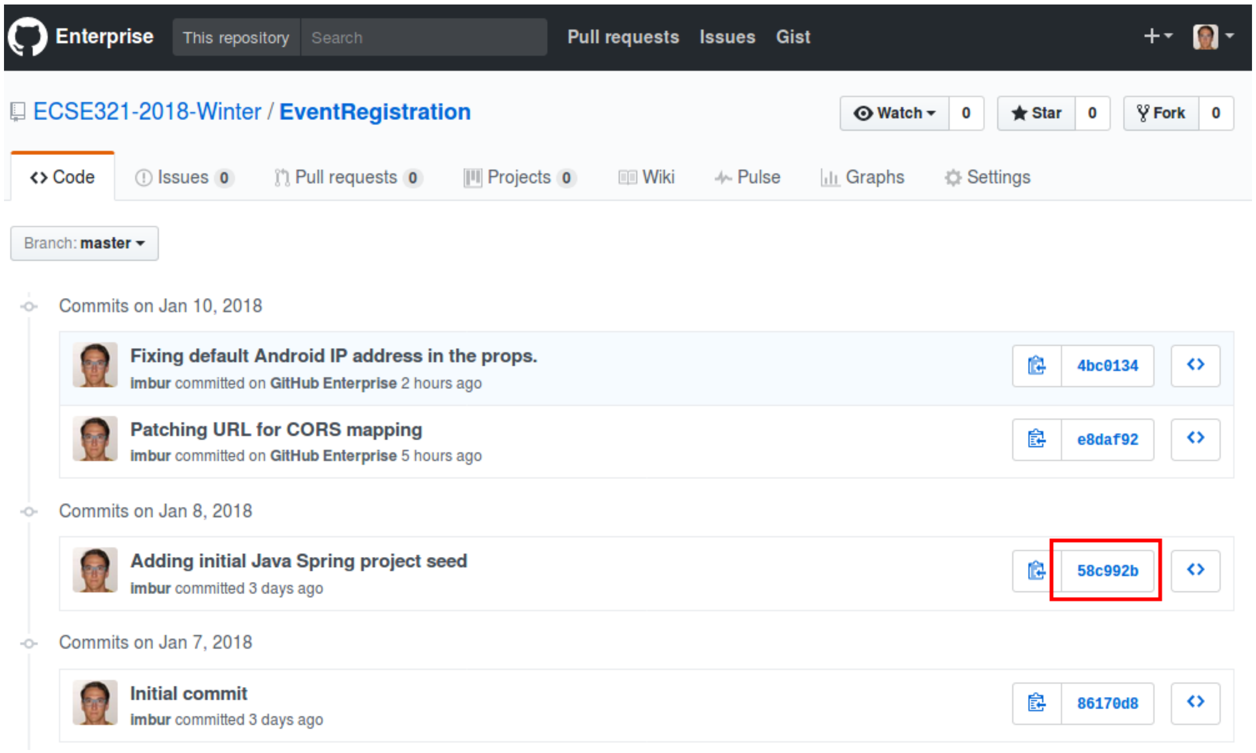
Project Reports: All project reports should be provided on the wiki page of the Github repository of your group. Clearly state the course name and number, term, team number, and team members on the overview page. Each deliverable should be on a different wiki page, formatted as a Markdown document.

Submission of Source Code: Your team is required to work on the deliverables in the repository assigned to your team in the GitHub organization of this course, but **a commit link** (i.e. the URL of your final commit) **is required to be submitted in myCourses**. For that purpose, you need to select the *commits* menu for a repository:



This repository contains the initial sources needed for the first assignment.

To **get a commit link** for a specific commit, click on the button with the first few characters of the hash of the commit and copy the URL.



Commits on Jan 10, 2018

Fixing default Android IP address in the props.
imbur committed on GitHub Enterprise 2 hours ago — 4bc0134

Patching URL for CORS mapping
imbur committed on GitHub Enterprise 5 hours ago — e8daf92

Commits on Jan 8, 2018

Adding initial Java Spring project seed
imbur committed 3 days ago — 58c992b

Commits on Jan 7, 2018

Initial commit
imbur committed 3 days ago — 86170d8

While it is recommended to use multiple branches in your GitHub repository, **you are required to integrate all features to the master branch when submitting a deliverable**. Contributions that exist on other branches will not be considered for grading.

Member Contributions: Each team member must contribute to each project deliverable. A team member who does not contribute to a project deliverable receives a mark of 0 (zero) for that deliverable. A team member may optionally email a confidential statement of work to the instructor ***before the due date*** of the project deliverable. A statement of work lists in point form how team members contributed to the project deliverables. In addition, the statement of work also describes whether the workload was distributed fairly evenly among the team members. A statement of work may be used to adjust the mark of a team member who is not contributing sufficiently to the project deliverable. It is not necessary to email a statement of work, if a team distributed the work for the project deliverable fairly evenly and each team member contributed sufficiently.

Note that contributions to your group's GitHub repository will be taken into account for the grade of some deliverables. You can view user activity on GitHub by opening your repository online and clicking on the Insights tab.