CS 246—Assignment 5, Group Project (Fall 2025)

Due Date 1: Friday, November 21st, 5:00pm
Due Date 2: Friday, November 28th, 11:59pm

This project is intended to be doable by three people in two weeks. Because the breadth of students' abilities in this course is quite wide, exactly what constitutes two weeks' worth of work for three students is difficult to nail down. Some groups will finish quickly; others won't finish at all. We will attempt to grade this assignment in a way that addresses both ends of the spectrum. You should be able to pass the assignment with only a modest portion of your program working. Then, if you want a higher mark, it will take more than a proportionally higher effort to achieve, the higher you go. A perfect score will require a complete implementation. If you finish the entire program early, you can add extra features for a few extra marks.

Above all, **MAKE SURE YOUR SUBMITTED PROGRAM COMPILES AND RUNS.** The markers do not have time to examine source code and give partial correctness marks for non-working programs. So, no matter what, even if your program doesn't work properly, make sure it at least does something. After submitting, download your code from Marmoset and make sure it compiles and runs.

1 The Game of Students of Watan

In this project, you will implement the video game Students of Watan, which is a variant of the game Settlers of Catan (often called Settlers for short), with the board being based on the University of Waterloo. If you are unfamiliar with Settlers, please see **here** for an overview. It may also be beneficial to play a game or two of Settlers to become familiar with gameplay but not necessary. Note that the rules of Watan vary from the rules of Settlers.

1.1 Gameplay Overview

During the setup of the game, each student selects starting achievements. Once the game begins, each student takes turns rolling the dice to determine which tiles produce resources. Everyone who has a criterion on the vertex of a tile matching the rolled number acquires the resource associated with the tile. After rolling, the student who rolled the dice has the opportunity to complete goals and criteria using their resources.

The goal of the game is to be the first of the four players to complete 10 course criteria.

2 Game Components

2.1 Board

The board (shown in Figure 1) is the visual representation of the state of the current game. The board will always display the 19 tiles (2.2) present in the game, all goals (2.5) and criteria (2.4), and the goals and criteria student have completed. The representation for each of these components is described in their respective sections.

2.2 Tiles

Each tile represents one type of resource in the game and is displayed as a hexagon on the board. The middle of the tile lists the type of resource, the value of a tile, and the location of the tile. In Figure 2, the location is 9, the resource is CAFFEINE and the value is 2. Additionally, if geese (2.3) are present on the tile, it will be displayed as in Figure 3. The outline of each tile is composed of the goals and achievements surrounding it.

There are 6 types of tile: CAFFEINE, LAB, LECTURE, STUDY, TUTORIAL, and NETFLIX. The Resources section (3.5) discusses how resources and values are distributed. NETFLIX is a tile that produces no resources[1].

2.3 Geese

A tile can be overrun with geese. In the event that a tile has been overrun by geese, students will not receive resources when the value matching the tile is rolled [2].

2.4 Course Criteria

A course criterion is located at the vertex of each tile. Each tile has 6 vertices. Some vertices are shared by multiple tiles.

In Figure 2, the criteria are numbers 20, 21, 26, 27, 32, and 33.

2.4.1 Criterion Completion

A *course criterion* can be completed. One student can complete each course criterion. Each criterion can have one of 3 completions on it. Once a student has a criterion completed, they can improve the criterion twice. Once completed, another student cannot have the same criterion. When a criterion is completed, the display is updated with the criterion number being replaced with the first character of the colour of the owner, e.g. 'B' for "Blue", followed by the first character of the name of the improvement, e.g. 'A' for "Assignment".

Completions must be bought in the following order:

1. Assignment: Completing an assignment gives the student one resource when the value of any adjacent tile is rolled. When acquired, an assignment gives the student one course criterion. An assignment costs one CAFFEINE, one LAB, one LECTURE, and one TUTORIAL.

2. Midterm: Completing a midterm gives the student two resources when the value of any adjacent tile is rolled. When acquired, a midterm gives the student one course criterion. A midterm costs two LECTURE, and three STUDY.

[1] Netflix is often a resource that prevents you from working on coursework

[2] Geese are terrifying creatures. Don't approach the geese.

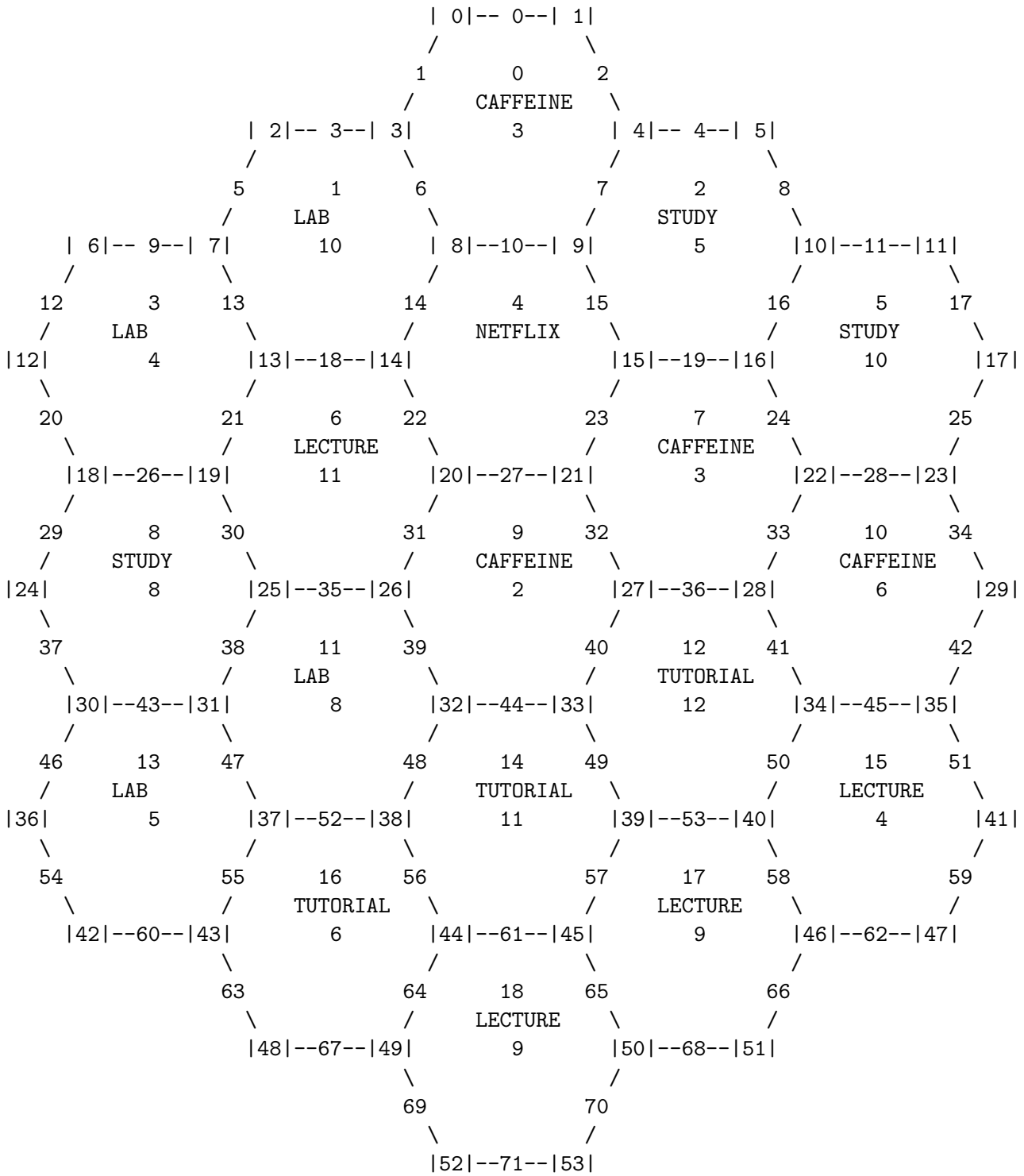

Figure 1: Sample Board

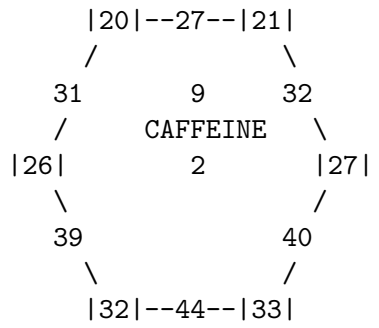

Figure 2: Sample Tile

3. Exam: Completing an exam gives the student three resources when the value of any adjacent tile is rolled. When acquired, an exam gives the student one course criterion. An exam costs three CAFFEIENE, two LAB, two LECTURE, one TUTORIAL and two STUDY.

This means you must complete a criterion first as an assignment, then a midterm, then an exam.

2.5 Goals

Goals are located at the edges of each tile going between two vertices. Each tile has 6 goals. Some goals are shared by multiple tiles.

For example, in Figure 2, the goals associated with the tile are numbers 27, 31, 32, 39, 40, and 44.

2.6 Goal Achievement

A *goal* can be achieved. Each goal can have one achievement placed on it. Once a student has achieved a goal, the goal cannot be achieved by another student. When a goal is achieved, the display is updated with the goal number being replaced with the first character of the colour of the owner, e.g. 'B' for "Blue", followed by 'A' (for "Achievement").

A student can achieve a goal if an adjacent criterion/goal has been completed/achieved by the student.

The cost of an achievement is one STUDY and one TUTORIAL resource.

2.7 Students

All students in the game are "human" players and controlled by the commands from standard input. Your game must support exactly 4 students.

Each student is assigned a colour at the beginning of the game with student 0 being Blue, student 1 being Red, student 2 being Orange and student 3 being Yellow.

When the status of a student is printed, output:

```
<colour> has <numCC> course criteria, <numCaffeines> caffeines, <numLabs> labs,
<numLectures> lectures, <numStudies> studies, and <numTutorials> tutorials.
```

When the completions of a student are printed, output:

`<colour> has completed:`

followed by the following line for each criterion completed:

`<criterionNumber> <criterionType>`

where `<criterionNumber>` is the number of each criterion that `<colour>` owns and `<criterionType>` is the number of the upgrade (1 for assignment, 2 for midterm, and 3 for exam). The criteria should be printed in the order they were originally completed.

2.8 Dice

Two types of dice are supported by the game: loaded[3] and fair dice. Each student has their own set of dice to roll. When dice are rolled, any student who completed criteria that are on a tile with the value matching the roll receives resources associated with the tile and the improvement(s) on the each intersection. When the game begins, all students will have fair dice.

For example, in Figure 2, if a student owned achievement 21 and the dice rolls were 1 and 1 (i.e. the total is 2), the student will receive a CAFFEINE resource.

2.8.1 Loaded Dice

When loaded dice are chosen, the student is prompted with the statement:

`Input a roll between 2 and 12:`

If the student's input is invalid, i.e. not an integer or out of the range, output the message:

`Invalid roll.`

followed by the previous prompt.

Once a valid roll has been entered, the entered roll is used for the turn.

2.8.2 Fair Dice

When fair dice are chosen, two numbers are randomly generated, each between 1 and 6. The sum of the two generated numbers are used for the turn.

3 Game Rules

An important part of the game is how each of the game components interact with each other. This section describes many of the rules explaining how pieces interact.

3.1 Completing Criteria

For a student to complete a criterion, the following two conditions must be met:

- No adjacent criterion is completed.
- It is either the beginning of the game, or they have achieved a goal that is adjacent to the criterion.

[3] Using loaded dice is a form of cheating in games. This is the only time we will encourage you to cheat in this course.

3.2 Improving Criteria

A criterion can be improved if the student has already completed the criterion and the criterion has not been improved to an exam yet. An assignment can only be improved by a midterm and a midterm can only be improved by an exam.

3.3 Achieving Goals

A goal can be achieved if the student has completed either an adjacent criterion or completed an adjacent goal.

3.4 Turn Order

Once the first criteria are chosen, it is the Blue student's turn, followed by Red, then Orange, then Yellow, then Blue, etc, until the end of the game.

3.5 Resources

3.5.1 Printing Resources

When printing owned resource types during the game, resources should always be printed in the order of CAFFEINE, LAB, LECTURE, STUDY, then TUTORIAL.

3.5.2 Obtaining Resources

When the dice roll is a non-seven number, students receive resources based on which criteria they have completed and which completion is on the criterion.

3.5.3 Using Resources

A student can use any resources that they have in their possession. Attempting to complete an action that requires more resources than the student has causes the action to fail and the message

```
You do not have enough resources.
```

to be printed.

3.5.4 Trading Resources

The active student can propose a trade with any other student. When a trade is proposed, the following output will be displayed:

```
<colour1> offers <colour2> one <resource1> for one <resource2>.
Does <colour2> accept this offer?
```

where `<colour1>` is the current student, `<colour2>` is the colour of the student `<colour1>` is proposing to trade with, `<resource1>` is the resource `<colour1>` is offering and `<resource2>` is the resource `<colour1>` is requesting.

After the prompt, player `<colour2>` responds. The response from `<colour2>` is "yes" or "no". If "yes", `<colour1>` gains one `<resource2>` and loses one `<resource1>`, `<colour2>` gains one `<resource1>` and loses one `<resource2>`. Otherwise, neither players' resources change.

3.5.5 Resource Distribution

The value and resource associated with each tile is randomly generated at the beginning of the game. The board consists of the following resources: 3 TUTORIAL, 3 STUDY, 4 CAFFEINE, 4 LAB, 4 LECTURE, and 1 NETFLIX. The board will consist of the following values: one 2, one 12, two 3-6, and two 8-11. The tile with NETFLIX does not display a value.

For example, suppose a 2 was rolled with the board displayed above. There is only one tile on the board that has the value 2 and it has the resource CAFFEINE. The criteria on the tile are criteria 20, 21, 26, 27, 32, 33. Each student who has completed one of these criteria receives the resource CAFFEINE. The number of CAFFEINE they receive is dependent on the type of completion on the criteria (one for assignment, two for midterm, and three for exam). If a student has completed multiple criteria on a single tile, they receive the resource for each criteria.

After the resources are distributed, print the following line for each student who gained resources, in the order of student number:

```
Student <colour> gained:
```

followed by the line:

```
<numResource> <resourceType>
```

for each resource they received from the dice roll.

If no students gained resources from the roll, output

```
No students gained resources.
```

3.6 Moving Geese

At the start of the game, the geese are not on the board. When a seven is rolled, the current student moves the geese to any tile on the board. Any student with 10 or more resources will lose half of their resources (rounded down) to the geese. The resources lost are chosen at random and are described below. For each student who loses resources, output:

```
Student <colour> loses <numResourcesLost> resources to the geese. They lose:
```

followed by the line:

```
<numResource> <resourceName>
```

for each resource lost.

The active student is prompted:

```
Choose where to place the GEESE.
```

The student responds with the tile number of any tile except where the GEESE is currently placed. The current student can then steal a random resource from any student who has completed a criterion on the tile to which they move the geese. The following is output:

```
Student <colour1> can choose to steal from [students].
```

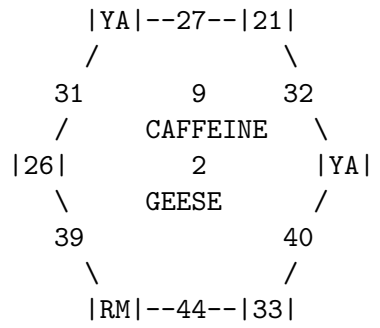

Figure 3: Tile with Geese

where `<colour1>` is the active student and `[students]` is each student, represented by their colour, who have achievements on the tile where the geese were placed, have a non-zero number of resources, and are not the active player, separated by commas. The students should be printed out in order of player number.

The active student is prompted:

```
Choose a student to steal from.
```

The active student responds with the colour of the student they want to steal from. After a resource is stolen, the following is output:

```
Student <colour1> steals <resource> from student <colour2>.
```

If the tile has no students who can be stolen from, the following is output:

```
Student <colour1> has no students to steal from.
```

The probability of stealing/losing each resource is the number of the resource the student being stolen from has, divided by the total number of resources the student being stolen from has. The display is updated with the word GEESE being added below the value of the tile that the geese were placed on. If the geese were previously on a tile, the word GEESE is removed from that tile.

After handling the geese, the active player completes their turn as normal.

For example, consider Figure 3. Say Blue has 5 STUDY, Red has no resources, Orange has 11 CAFFEINE, and YELLOW has 3 LECTURE. Now Blue rolls a 7. The program outputs:

```
Student Orange loses 5 resources to the geese. They lose:
5 CAFFEINE
Choose where to place the GEESE.
```

Blue inputs 9 to place the GEESE on tile 9. The program outputs:

```
Student Blue can choose to steal from Yellow.
Choose a student to steal from.
```

Blue inputs Yellow. The program outputs:

```
Student Blue steals LECTURE from student Yellow.
```

Now Blue has 1 LECTURE, 5 STUDY, Red has no resources, Orange has 6 CAFFEINE, and YELLOW has 2 LECTURE.

3.7 Saving a Game

When a game is saved, the following information is printed to a file in the following order:

```
<curTurn>
<student0Data>
<student1Data>
<student2Data>
<student3Data>
<board>
<geese>
```

The `curTurn` is the turn of the first student that should roll the dice when the game is loaded. If the dice have been rolled on during the turn when the save is called, than `curTurn` is the value of the student after the current student.

A student's data is printed out as follows:

```
<numCaffeines> <numLabs> <numLectures> <numStudies> <numTutorials> g <goals> c <criteria>
```

where `<goals>` is the number of each goal completed, separated by one space, and `<criteria>` is the list of criteria met where each pair of numbers is the criterion number followed by the number of the completion on it.

For example, the row

```
1 2 1 2 3 g 16 36 19 c 10 1 15 3 27 2
```

could be the output for a player with 1 CAFFEINEs, 2 LABs, 1 LECTURE, 2 STUDYs, 3 TUTORIALs, the goals 16, 36 and 19 achieved and the criteria 10 being an assignment, 15 being an exam and 27 being a midterm.

The `<board>` will be the order of the resources on the board with 0 representing CAFFEINE, 1 representing LAB, 2 representing LECTURE, 3 representing STUDY, 4 representing TUTORIAL, and 5 representing NETFLIX with each each resource being followed by its value

For example, the board in Figure 1 is

```
0 3 1 10 3 5 1 4 5 7 3 10 2 11 0 3 3 8 0 2 0 6 1 8 4 12 1 5 4 11 2 4 4 6 2 9 2 9
```

Note that the resource for NETFLIX will always be followed by the number 7 even though it doesn't have a value associated.

The `<geese>` is a number between 0 and 18 representing the tile that contains the geese.

4 Gameplay

There are four distinct stages in the game, during which input should be handled differently.

4.1 Beginning of Game

At the beginning of the game, each student chooses two initial assignments to complete. The assignments will be chosen by students in the order Blue, Red, Orange, Yellow, Yellow, Orange, Red, Blue.

Each student is prompted with the statement:

`Student <colour>, where do you want to complete an Assignment?`

Each student responds with the number of a valid intersection.

Once a valid intersection has been entered, prompt the next student. Once all students have placed their first assignments, print the updated board and begin the game.

4.2 Beginning of Turn

At the beginning of a turn, print the following:

`Student <colour>'s turn.`

followed by the status of the student. A student can then enter any of the following three commands:

- `load`: sets the dice of the current student to be loaded dice
- `fair`: sets the dice of the current student to be fair dice
- `roll`: rolls the current student's dice

4.3 End of Turn

At the end of a turn, a student can input any of the following commands:

- `board`: prints the current board
- `status`: prints the current status of all students in order from student 0 to 3
- `criteria`: prints the criteria the current student has currently completed
- `achieve <goal\#>`: attempts to achieve the goal at `<goal\#>`
- `complete <criterion\#>`: attempts to complete the criterion at `<criterion\#>`
- `improve <criterion\#>`: attempts to improve the criterion at `<criterion\#>`
- `trade <colour> <give> <take>`: attempts to trade with student `<colour>` giving one resource of type `<give>` and receiving one resource of type `<take>`
- `next`: passes control onto the next student in the game.
- `save <file>`: saves the current game state to `<file>`
- `help`: prints out the list of commands

The help command prints the following

```
Valid commands:
board
status
criteria
achieve <goal>
complete <criterion>
```

```
improve <criterion>
trade <colour> <give> <take>
next
save <file>
help
```

4.4 End of Game

The game is complete when a student has a total of 10 completed course criteria [4]. The first student to have 10 completed course criteria is the winner of the game. At this point, the students are prompted:

```
Would you like to play again?
```

If the response is "yes", the game starts over from the beginning. If the response is "no", the game exits.

4.5 Handling Input

When waiting for a response from cin, the program will print: >.

If a student attempts to improve or complete an achievement, or complete a goal, and the space is invalid (for any reason), output:

```
You cannot build here.
```

If a student attempts to build anything and they do not have the required resources, output:

```
You do not have enough resources.
```

If an unrecognized command is entered, output:

```
Invalid command.
```

If the game receives an end-of-file signal at any point after initial setup, the game ends with no winner, and saves the current game state to `backup.sv`.

5 Command-line Interface

Note: Command line options are important to implement as they aid in easy testing of the correctness of your program. Programs which do not implement all command line options will not be fully tested and will receive low marks.

Your program must support the following options on the command line:

- `-seed xxx` sets the random number generator's seed to `xxx`. If you don't set the seed, you always get the same random sequence every time you run the program. It's good for testing, but not much fun.

[4] Unlike in real courses, in Watan, you decided the mark distribution for the course

- `-load xxx` loads the game saved in file `xxx`. You may assume that the file being loaded was saved as a valid game state as described in the Saving section. **Note: loading from a saved file will be heavily used during the testing of the project and should be considered a priority while implementing the project.**
- `-board xxx` loads the game with the board specified in the file `xxx` instead of using random generation. This file will contain the order of the tiles for the game as described in the saving the game function. The number of each tile present does not have to match the random generation. e.g. It is valid to have a board which is entirely TUTORIAL resources with the value 12[5].

The command-line options can be given in any order.
Note: only one of `-board` and `-load` will be provided at a time.

6 Questions

You must answer questions 1 and 2, AND any 3 of questions 3-7. We recommend reading and thinking about these questions while you are designing you project.

1. You have to implement the ability to choose between randomly setting up the resources of the board and reading the resources used from a file at runtime. What design pattern could you use to implement this feature? Did you use this design pattern? Why or why not?
2. You must be able to switch between loaded and unloaded dice at run-time. What design pattern could you use to implement this feature? Did you use this design pattern? Why or why not?
3. We have defined the game of Watan to have a specific board layout and size. Suppose we wanted to have different game modes (e.g. square tiles, graphical display, different sized board for a different numbers of players). What design pattern would you consider using for all of these ideas?
4. At the moment, all Watan players are humans. However, it would be nice to be able to allow a human player to quit and be replaced by a computer player. If we wanted to ensure that all player types alway followed a legal sequence of actions during their turn, what design pattern would you use? Explain your choice.
5. What design pattern would you use to allow the dynamic change of computer players, so that your game could support computer players that used different strategies, and were progressively more advanced/smarter/aggressive?
6. Suppose we wanted to add a feature to change the tiles' production once the game has begun. For example, being able to improve a tile so that multiple types of resources can be obtained from the tile, or reduce the quantity of resources produced by the tile over time. What design pattern(s) could you use to facilitate this ability?
7. Did you use any exceptions in your project? If so, where did you use them and why? If not, give an example of a place that it would make sense to use exceptions in your project and explain why you didn't use them.

[5]This would lead to a very boring game.

Grading

Your project will be graded as described in the project guidelines document.

Even if your program doesn't work at all, you can still earn a lot of marks through good documentation and design, (in the latter case, there needs to be enough code present to make a reasonable assessment).

Above all, make sure your submitted program runs, and does something! You don't get correctness marks for something you can't show, but if your program at least does something that looks like the beginnings of the game, there may be some marks available for that.

7 If Things Go Wrong

If you run into trouble and find yourself unable to complete the entire assignment, please do your best to submit something that works, even if it doesn't solve the entire assignment. For example:

- only one criterion completion exists
- does not recognize the full command syntax
- course criteria is not calculated correctly

Note that for testing purposes, it is more important to have the non-randomized features, e.g. loaded dice, reading a saved game from a file, working than the randomized features, e.g. fair dice, random generation.

You will get a higher mark for fully implementing some of the requirements than for a program that attempts all of the requirements, but doesn't run.

Every time you add a new feature to your program, make sure it runs, before adding anything else. That way, you always have a working program to submit. The worst thing you could do would be to write the whole program, and then compile and test it at the end, hoping it works.

A well-documented, well-designed program that has all of the deficiencies listed above, but still runs, can still potentially earn a passing grade.

Plan for the Worst

Even the best programmers have bad days, and the simplest pointer error can take hours to track down. So be sure to have a plan of attack in place that maximizes your chances of always at least having a working program to submit. Prioritize your goals to maximize what you can demonstrate at any given time. We suggest: save any graphics for last, and first do the game in pure text. One of the first things you should probably do is write code to draw the display (probably a class with an overloaded friend `operator<<`). It can start out blank, and become more sophisticated as you add features. You should also do the command interpreter early, so that you can interact with your program. You can then add commands one-by-one, and separately work on supporting the full command syntax. Take the time to work on a test suite at the same time as you are writing your project. Although we are not asking you to submit a test suite, having one on hand will speed up the process of verifying your implementation.

You will be asked to submit a plan, with projected completion dates and divided responsibilities, as part of your documentation for Due Date 1.

If Things Go Well

If you complete the entire project, you can earn up to 10% extra credit for implementing extra features. See **guidelines.pdf** for more details.

Submission Instructions

See **guidelines.pdf** for **important** instructions about what should be included in your Due Date 1 and Due Date 2 submissions. Read it now!