Capture the Flag Project

Due August 15 or 16 at your Demo time.

Capture the Flag

In this project you will implement the game Capture the Flag. If you are unfamiliar with the game, you can read about it here: `http://en.wikipedia.org/wiki/Capture_the_flag`.

There are many variants of the game (as you can see from the wiki document). For our game, the rules will be defined below in this project document.

1: The Basic Game Rules

Capture the flag is a game played by two teams, which we'll refer to as team 1 (**Blue** team) and team 2 (**Red** team), on a playing field which is divided into two territories.

Each team in the game has

- a "territory" on the playing field (side 1 or side 2),
- a "base" that is located somewhere in their territory,
- a "jail" that is located somewhere in their territory,
- six or more players, and
- a "flag" (that is initially located at their base).

The object of each team is to capture the opposing team's flag and bring it back to their base. The first team to do this wins the game.

There are several notions of being "caught" in the game that depend on where a player is and if they have the flag or not. Being caught essentially means an opposing player is "close enough" to them to catch them.

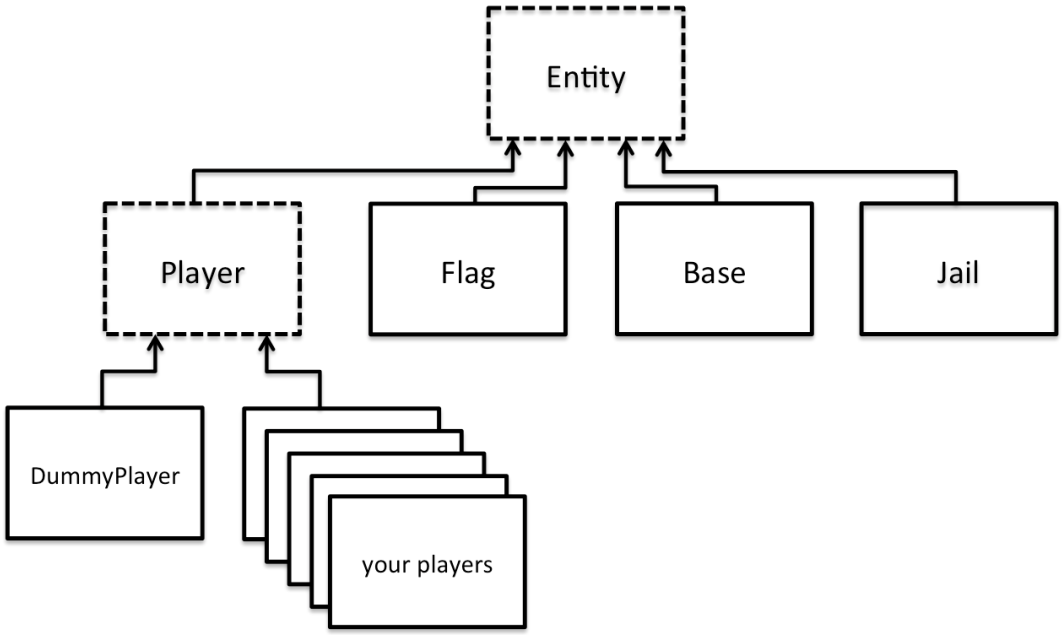
- When a player is in the other team's territory and they are caught by an opposing player they are taken to the other team's jail. If that player is also holding the other team's flag then the player is taken to jail and the flag is dropped where it is.
- When a player has the other team's flag and is in their own territory (trying to bring the flag to their base) and they are caught by an opposing player the flag is returned to the other team's base and the player resumes its actions. The player is not sent to jail.
- When a player is in their own territory and is not holding the other team's flag they cannot be caught.

When a player is caught and sent to jail they will remain in jail until another player on their team, who is not in or being taken to jail, "frees" them. A player frees a teammate by being "close enough" to them.

2: Classes in the Game

There are several classes involved in the game. Some are provided and should not be changed (alter at your own risk!), some are provided and need to be completed, and some you will need to create.

The "things" in the game all extend an abstract `Entity` class.



Entity: This abstract class provides the base for "things" in the game such as players, flags, jails and bases.

Player: A direct child of Entity, this abstract class contains the basic framework for all players in the game.

DummyPlayer: A provided concrete player that doesn't do much. It is used to get the game up and running as a starting point.

Your own Players: You will implement several different kinds of Players that behave differently in the game.

Flag: A direct child of Entity, this is for the two flags in the game. They will not do much in a basic game (just keep a location).

Jail: A direct child of Entity, this is for the two jails in the game. They will not do much in the basic game (just keep a location).

Base: A direct child of Entity, this for the two the bases in the game. They will not do much in the basic game (just keep a location).

Team: You will create a `Team` class that holds a team of players.

Field: This class provides the base for the playing field and for overseeing the actual game.

View, SpriteStore, Sprite: These classes provide graphics for the game. You do not need to modify these.

CaptureTheFlag: This is the game itself. It drives the game (with a `main` method).

EntityOutOfBoundsException: This is an exception that will be thrown whenever a player leaves the field.

In Assignment 5, you are asked to write some player classes that have certain behaviour. These classes will lead you along to implementing the players you will need for the project.

> There is a lot of different things that need to be done to complete the game. Do not be discouraged by this. Implement whatever you can for the game (as described below). Spread the work out in your team. Be sure to test each other's contributions. You will find that it is easier to test someone else's code than your own.

3: The Players

In a game of capture the flag each team will have six or more players. You will implement five different behaviours for them. In particular, you will need to implement the following.

- Players that try to capture the opponents flag and bring it back to their home (to win)
- Players that try to catch opponents (so that they go to jail)
- Players that try to free their teammates from jail
- Players that are defenders of their base (trying to catch opposing players)
- Players that try to defend against (catch) flag carriers

All the players must stay in the playing field at all times. You will throw an `EntityOutOfBoundsException` object when a player leaves the field.

4: The Team Class

Write a `Team` class that has a constructor

```
public Team(Field theField, int numberOfPlayers)
```

You can add other constructors if you wish. The purpose of the Team class is, as the name suggests, to represent teams in your game. The intention is that your `CaptureTheFlag` game will have code similar to

```
Team reds   = New Team(field, 24);
Team blues  = New Team(field, 24);
```

which creates, initializes and registers all players that will be playing in the game. This will replace the `for` loop that creates players in the `main` method of the provided CaptureTheFlag.java file.

You'll have to make some decisions about what to do with Team. In a basic implementation, it should be able to create different teams for the game. You may add any new constructors and methods as needed.

An important component of the project is that you must present your code to be evaluated while it is running. In order to do this, you need to run the game with different scenarios so that different (needed) behaviour of your players is observable. In order to do this, your Team class should be able to create the following teams:

- A full team with at least one of each of the required types of players (from above). This will be used for playing the game.
- A team with any number of different players. For example, you should be able to easily create a team with a single player that tries to capture the other team's flag. Such a team is useful so that we can easily test each different kind of player.

If your project team has not completed all the different players, just include all of those that you finished. If you do not finish the Team class, be sure that your submission allows for easy demonstration of each different player that is completed.

For full marks for the `Team` class, you will need to have methods that allow for reading a team from a text file. The file should include all player information and also team stats like how many games they have played and how many wins/losses. After a game is completed, you should update (rewrite) the team file with the updated stats.

5: The Field Class

Finish the incomplete methods in the `Feild` class so that your players can play the game. In particular, you should write code for or modify the existing code in

- `winGame`
- `freeTeammate`
- `pickUpFlag`
- `catchOpponent` This code is not robust but works.

There are several (empty) methods in the `Player` class that you can implement to help you with your implementation of the Field methods. (The methods should provide some functionality to allow a player to know, for example, if it has the flag or if it has been caught. Add any other methods, like `isInJail` or `hasFlag`, as you need.)

6: Extending the Game

The project is worth 10% of your final grade. You can receive bonus marks by extending the game beyond what is asked for here.

Extensions include, but are not limited to

- Add other Entity classes to the game; rocks, trees and other obstacles. Do not allow the players to walk through the obstacles.
- Adding new players with logic for players that go beyond the basic behaviour described here. For example, a player that tries to capture the flag will not go directly to the flag but might try and go around and behind it to avoid being caught.
- Having a player change teams after they have been caught twice.
- Limiting the "vision" of the players. For example, each player might only be able to see things in a given radius around them. A `hunter` player would be needed to find the flag.
- Allow some players to run fast for a while, but then get tired and move slower than normal.
- Have messages displayed on the bottom message window of the View that gives a play-by-play of the game. (That is, it displays messages when things happen in the game. For example, "Cat from team ravens picked up the flag", "Dog from team geegees went out of bound", "Akhil from team geegees was caught and sent to jail by Stanley".)
- Make the message (above) scroll across the message window.
- Anything that you think would be cool to add to the game.

Note: You do not need to "complete" the basic game to add extras. You can still receive extra marks as long as the extra thing you are adding works and we can be visually confirm them in your presentation. Marks will be based on the difficulty of the extra added.

Note: The files as provided will not compile. You will have to fix the bug(s) in order to get it up and running. This is the first task of the Assignment 5.

7: Evaluation

Your grade for the project will come from three different components

- The quality of the code presented. (Team mark)
- Your answers to questions during the demonstration. (Individual mark)
- Peer evaluation. (Individual mark)

If everyone in the team contributed, roughly, the same to the project then your grade will be the project demo mark. If it is deemed that you did not contribute enough to the project, your mark will based on the demo mark and the projected contribution you made. Those that contribute very little to the project will receive very little marks (fail).

Project Submission

Your team should submit your project in a file called `projectTeamName.zip`, where TeamName is your team name. Your project should only be submitted once for your entire team.

Everyone will also submit a peer evaluation form (.txt file) to the appropriate place in cuLearn.