Project Checkpoint 1: Game State

CISC181 Fall 2013
Assigned: Oct 2
Due: Oct 9 at 11:55PM on Sakai
Text references: Chapters 5, 6, 7, 8, 10, 11

* Problems you need to solve and turn in are listed in green text.

Project Checkpoint 1

In the previous Bioinformatics exercise, you created a model (a dot matrix) and a scoring mechanism (maximum consecutive sequence alignment) for a game (figuring out which was the most likely source location of an influenza strain). While we may not think of finding the best treatment options for a new strain of influenza as a game, the strategy we used can be readily applied to many games.

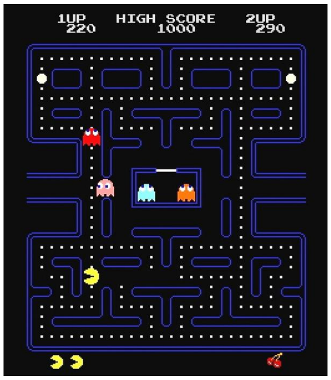
In many games, the basic model can be represented as a "board" and a collection of pieces.

Examples of games:

- Pacman
- Bejeweled
- Minesweeper
- Breakout
- Tetris
- Escape
- Tower Defense
- Centipede
- Scrabble
- Carcassonne
- Candy Land
- Angry Birds

Off-limits games:

- Tic-tac-toe, Snake, Othello, Connect4



In addition to a basic model (which would include a structure for the board, and classes for data and game piece objects), each game has a set of rules encoded as moves that are valid for various states of the game. The game also has a way to determine if a player is "winning" and/or how successful each player is at playing the game.

In this first checkpoint, you must submit to Sakai:

1. An updated Project idea document that includes answers in English to the "back page" of the idea form (see the last page of this document)
2. Your exported game project from Eclipse that includes:
   - A new package for all of your code that matches your UD username (choose one of your 2 partner's usernames) -- you need to use the convention edu.udel.username.gamename
   - Classes for your data model
   - A test class that creates various states of your game using objects from your data model. At a minimum, you should include a starting state, an ending state, and a mid-game state.
   - A state.toString method that will show a very basic text output for a state in your game model
   - A main method in your test class that prints various states
3. A memory diagram showing the contents in memory for a single mid-game state.

See the next few pages for examples provided for TicTacToe5x5 and Snake.

NOTE: There should be no state mutation in your code at this point. Your game should NOT work yet, nor should it have any of the game logic that makes it "do" anything.

Here is an example describing in English two game variations:

TicTacToe5x5
Modifications:

- using a 5x5 grid instead of 3x3
- the middle square is filled, neither player can use it

Scoring:

- the winner connects any 4 adjacent squares
- the game ends when there is a winner or all of the squares are filled
- the heuristic (the guess as to how close a player is to winning) is simply the maximum number of any adjacent squares a player has
    - each player then has a heuristic score of 0 before the game starts, 1 after they have placed their first marker, and the winner will have a score of 4

Snake
Modifications:

- not much really, this is a basic version
- snake starts at length 2, grows until length 13 by 1 each time food is eaten
- at length 13 a new level is reached
    - each level snake goes back to length 2 and speeds up by 20%

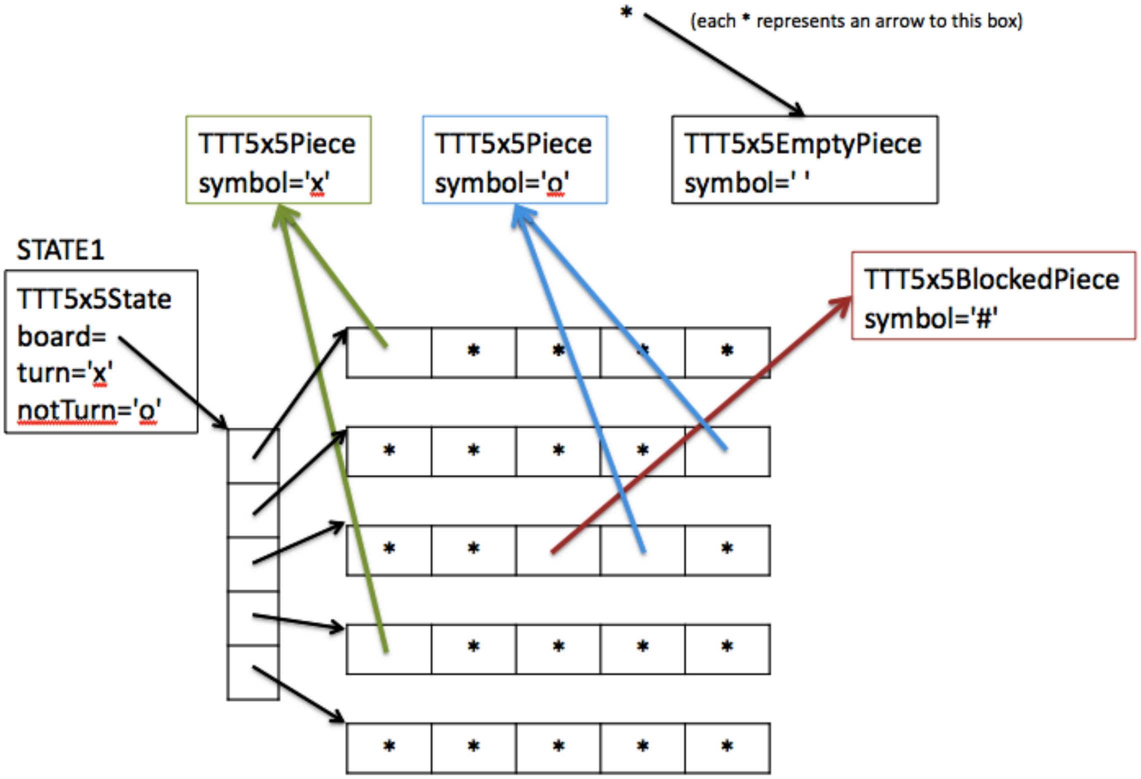
Scoring:

- there is no winner (single player game), but there is a high score record
- the game ends when snake runs into itself or boundary
- the player score increases by 1 for each food eaten
- the heuristic (the guess as to how close a player is to winning) is simply the current score + how close the snake is to the food

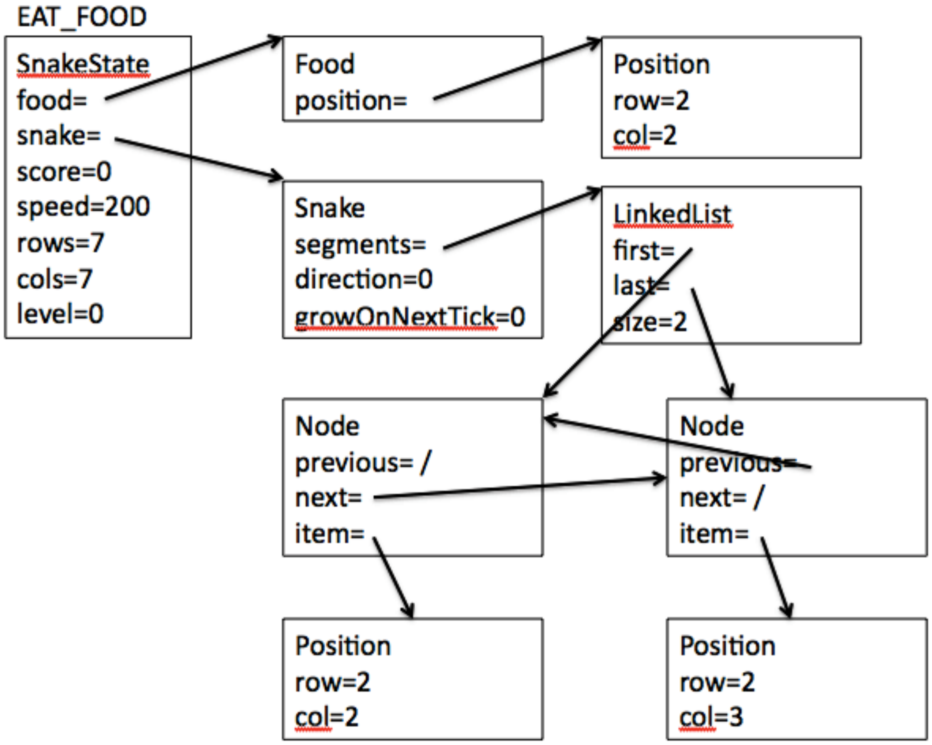
Code with example board setups is included in CP1_tictactoe.zip and CP1_snake.zip. You can run the test to see some output.

A memory diagram for the given TicTacToe5x5 state and the Snake state are on the following page.

TicTacToe5x5:



Snake:



With your partner, complete these questions.

Game Idea Details (be concise here, you will turn in a more detailed write-up next week)

Team member names:

Game Title:

Initial Model (classes, properties, data structures?):

Significant modifications, how is your game different than the original game rules / model?:

Scoring rules: