**Concurrent Programming**

**Assignment SP2 2017**

**Written By**

**Travis McDermott**

**Introduction**

This document defines the direction taken, to complete the application of a multiplayer snakes’ game. Below you find constraints and assumptions defined by the group. Classes with detailed definitions and structure explaining the implementation can as well be found.

**Constraints**

1. Sever will have the game state.
2. Player should be able to login in.
3. One multiplayer game for everyone to join.
4. Game is called multiplayer snakes.
5. Players send moves to the server.
6. Move are up, down, left and right.
7. Moves are collected from server then the game board is updated.
8. Once fruit has been eaten, the player scores are updated
9. No Networking
10. Players and Server to be threads on Java Virtual Machine
11. Server to draw game state.
12. Game must be played by 4 real players
13. Game must be able to simulate up to 100 simulated players.
14. Simulated players make random moves.

**Assumptions**

1. Walls cannot be passed through.
2. Players can pass through own self.
3. If no inputs are passed snake continues in forward direction.
4. There will be a timer for players to make decision on direction. This timer will be a very small amount of time to keep game moving.
5. The timer will stay at constant speed.
6. Once a player make a collision with the wall or another player the player is removed from the current game and score is updated to leader board.
7. Snakes will grow to an infinite size, which will increase the game difficulty over time.
8. Fruit will be placed on board for players to eat and gain points after a certain random time.
9. Snakes will start at a length of 4.
10. Fruit cannot be place over an existing player of another piece of fruit.
11. Fruit will last on screen until eaten.
12. Only one player can eat the fruit.
13. Fruit will only occupy one unit of space.
14. The last place player will gain extra points for winning the game.

**Classes**

**Server**

The server will hold and create the game state. It will hold and array of threads that are used by the players and when not in use will be stored in a blocking queue. This will be decided by a Boolean to determine if the thread has stopped. This will increase efficiency allowing the reuse of the player thread.

The Server constructor will pass two parameters, the number of threads and the maximum possible number of tasks. These parameters will set the size of the array list of threads and the size of the blocking queue. From the constructor, the threads are executed.

The Server will have a synchronized method of execute and another synchronised method stop. Execute will add the task to the blocking queue, which will interrupt the current thread. This method will throw an error if the thread is already stopped. The stop method will stop all threads in the thread pool.

The Main method in server will wait for the users input until a set time and execute the players command. This will then simulate the world/position of the players and then broadcast this too all players.

**Player**

The player class will implement the runnable interface. The player class will send input to server.

**BlockingQueue**

The blocking queue will consist of one data structure and one variable. The data structure will be a linked list to resemble a queue. The variable will be the limit of the blocking queue, which in this case will be 100 simulated players by default.

**SnakesGamePlay**

**LTSA modelling tool**