

Multiplayer Snake

Software Engineering 1 & Intro to Java

by
Ian Laird & Andrew Walker

prepared for
Dr. Tomas Cerny

April 24, 2018

1 Outline

This game implements a 2 player version of the popular game Snake. Communication between the host and client is done using TCP. Players can pick up power ups and kill the other player to increase their score. The top 10 high scores are stored in a file.

2 Vision Statement

The next step in this game will be making it so that more than 2 players can play at the same time.

3 Gantt

4 Requirements

- 1.

5 Business Rules

1. Client must know the ip address of the server.
2. Each game must have two players.
3. High scores are remembered.

6 Use Cases

6.1 Use Case 1

Name: UC1-Initialize game

Primary Actor: Local user

Secondary Actor: Secondary user

Trigger: Game is created

Main Success Scenario

- 1 Primary and secondary users both want to play the game.
- 2 Network connection is established to secondary player.
- 3 Snakes and Tokens are generated with random locations.
- 4 Screen is created.
- 5 Game is now ready to be played.

Alternate Scenarios

- 1a There is no secondary player to play with the primary player.
 - 1 System will wait until another player connects or user closes the application.
- 2a network connection fails to establish.
 - 1 System will inform the user and then terminate.
- 3a There is a collision where random locations are identical or very close to each other.
 - 1 Snake and Token Locations will be recalculated.
 - 2 This process will be repeated until all positions are distinct and valid.

6.2 Use Case 2

Name: UC2–Play Game

Primary Actor: Local user

Secondary Actor: Secondary user

Pre-condition: Game is created

Main Success Scenario

- 1 Primary user makes and sends move.
- 2 Primary user receives secondary users move.
- 3 Each snake is replotted with the new position as the head.

Alternate Scenarios

- a Connection to other player is lost.
 - 1 System will notify player and terminate.

3a Player is dead after move.

- 1 Defeat screen is displayed.
- 2 Both players choose to play again and play resumes.

3b Secondary player is dead after move.

- 1 Victory screen is displayed.
- 2 Both players may choose to play again.

3ab.2a Player chooses to not play again.

- 1 High scores are updated.
- 2 Game terminates.

3ab.2b Player 2 chooses to not play again.

- 1 High scores are updated.
- 2 Game terminates.

6.3 Use Case 3

Name: UC3–play Again

Primary Actor: Local user

Secondary Actor: Secondary user

Pre-condition: Game over

Post-condition: Game is not over

Main Success Scenario

- 1 Snakes and Tokens are generated with random locations.
- 2 Screen is redrawn.

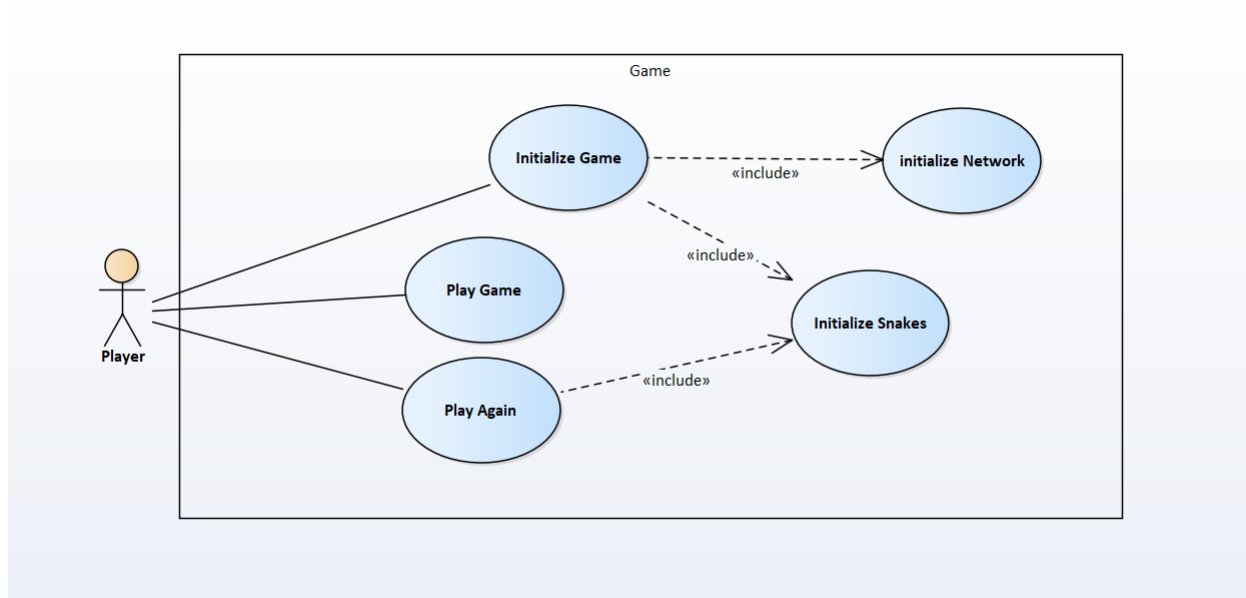
Alternate Scenarios

- a Connection to other player is lost.
 - 1 System will notify player and terminate.

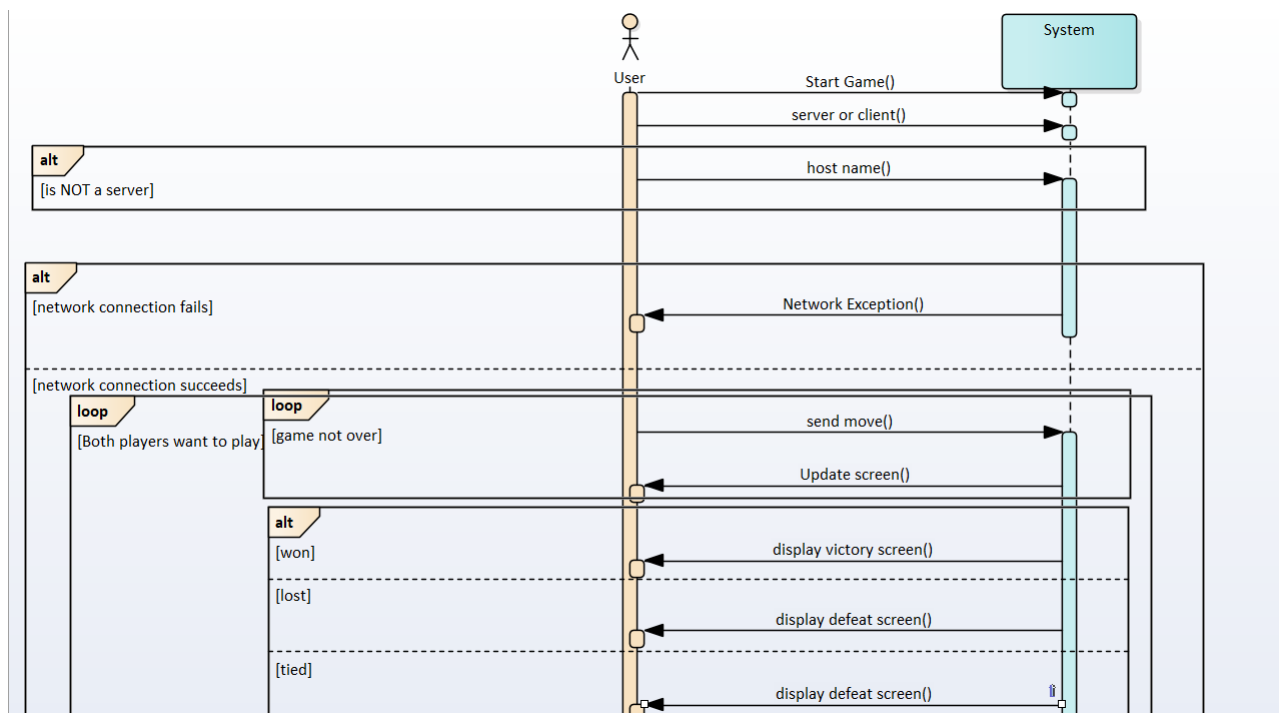
1.a Tokens and Snakes do not have independent positions.

- 1 System will recalculate positions.
- 2 This will be repeated until all positions are independent of each other.

7 Use Case Diagram



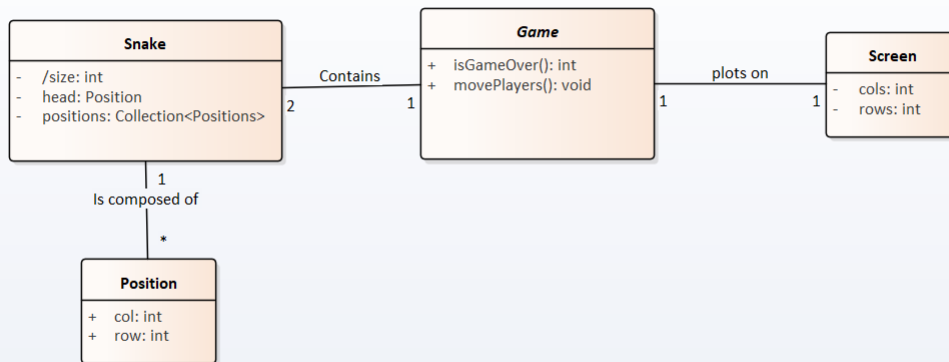
8 System Sequence Diagram



9 System Operations

1. *initGame()*
2. *server_or_client(boolean mode)*
3. *sendMove(Direction move)*

10 Domain Model



11 Operational Contracts

11.1 Contract 1

Operation: initGame
Cross References: UC1
Pre-conditions: No instance of a Game exists
Post-conditions:

- A Game is initialized.

11.2 Contract 2

Operation: server_or_client(boolean status)
Cross References: UC1
Pre-conditions: Game network has not been initialized previously.
Post-conditions:

- A socket has been initialized and connected to.
- If client, a connection will be made to server.
- If server, a client will be accepted.

11.3 Contract 3

Operation: sendMove(Direction move)
Cross References: UC2 and UC3
Pre-conditions: Game has been initialized, and both players are still alive.
Post-conditions:

- Player one and player 2 are moved to new cells.
- If player does not make move, their last move will be repeated.
- If the received moves kill a player, that player will die. The other player will receive points.

12 Sequence Diagrams

13 Design model

14 GRASP

15 Design Patterns

1. **Singleton:**

Singleton was the most used design pattern in this project. Many of the objects used lent themselves to only letting one instance exist at any time. The game screen and the game itself were both singletons.

2. **Factory method:**

Factory method was used to remove the need to use new to create instances of a Snake. The Cell method also had a getRandom method that functioned much like a factory.

3. **Abstract Factory:**

Abstract factory was used to create instances of a Game Object. Because there were multiple subtypes of the Game Object the abstract factory made it simpler to create an instance. Also the abstract factory allowed only one instance of any of game's subtypes to exist.

4. **Builder:**

A Builder is used to create instances of a Snake. Because the snake is a complex object builder lets the application customize them. It allows Snake location and color to be both be customized.

5. **State:**

State is used in the Screen to represent movement direction as determined by user keystrokes. It uses polymorphism to avoid using a switch statement in calculating the new snake position.

16 Stats

16.1 Hours Spent per User

Ian Laird: 20 hours

Andrew Walker: 10 hours

16.2 Logical Lines from Each User

Ian Laird: lines

Andrew Walker: lines