SuperSnake: Make Snake Game More Dynamic

박준영

2025. 6. 18.

1 Goal

배경 뱀 게임*Snake Game*은 간단한 규칙과 중독성 있는 매력으로 오랜 시간 사람들에게 사랑받아온 고전 비디오 게임 가운데 하나이다. 1976년 그렘린 인더스트리스*Gremlin Industries*사에서 블록케이드*Blockade*라는 이름으로 출시한 것이 최초이며, 이후 많은 발전을 거쳐 오늘날에 이르고 있다. 구글에서 제작한 뱀 게임도 있으며, 구글에 '스네이크 게임'을 검색하면 실행해 볼 수 있다.

목표 웹 환경에서 실행할 수 있는 가속 뱀 게임, 이름하여 SuperSnake를 제작하는 것이 이 프로젝트의 목표이다. 뱀 게임은 전 세계의 사람들에게 오랜 시간 사랑받아온 고전 게임이지만, 규칙의 단조로움 때문에 게임 진행이 길어질수록 사용자가 지루해지기 쉽다. 이에, 게임 진행 경과에 따라 뱀의 이동 속력이 높아지도록 하여 기존 뱀 게임의 한계를 보완한 '가속 뱀 게임'을 제작하였다.

뱀 게임의 특성을 고려할 때, 데스크톱 앱보다는 웹 앱으로 구현하는 것이 적합하다고 생각하였다. 따라서 뱀 게임에 맞는 웹 GUI를 구현하고, 세부적인 게임 로직은 자바*Java*로 작성하였다.

2 Requirements

2.1 Requirements

요구사항 프로젝트의 요구사항을 정리하면 다음과 같다.

- 일정 주기마다 뱀의 위치를 정해진 방향으로 이동시키기
- 방향키를 입력받아 뱀의 진행 방향을 조작하기
- 뱀이 사과를 먹으면 점수를 1점 올리고 뱀 몸통 길이를 1칸 늘리기
- 화면 위 무작위의 칸에 사과를 생성하기
- 뱀이 장애물에 부딪히면 게임을 끝내고 점수를 기록하기

이들 요구사항을 효율적으로 관리하기 위해 게임 상태, 진행 방향, 뱀의 위치 등을 적절히 묶어 객체로 정의하고 관리하였다.

2.2 Tools & Targets

개발 환경 이 프로젝트는 백엔드 개발과 프론트엔드 개발로 나누어 진행하였다. 프론트엔드는 리스크립트*ReScript*와 리액트*React*를 사용하여 구현하였다. 웹 GUI를 구현하기 위해서는 자바스크립트 계열의 언어를 사용하는 것이 효과적이고, 그중에서도 안전한 타입 시스템을 제공하는 리스크립트를 프론트엔드 언어로 채택하였다. 리액트는 자바스크립트 계열 언어에서 사용할 수 있는 라이브러리로, 상태를 실시간으로 관리하는 등의 기능을 제공한다.

한편, 백엔드는 '컴퓨터프로그래밍' 강의를 통해 학습한 자바를 사용하여 구현하였다. 뱀 게임에는 뱀, 보드 등 다양한 객체가 존재하므로, 자바의 객체지향적인 특성을 잘 활용하면 게임을 효율적으로 구현할 수 있을 것이라 판단하였다. 자바린*Javalin*은 자바로 작성된 웹 프레임워크로, 통신을 가볍고 빠르게 구현할 수 있게 해준다. 따라서 이 둘을 조합하여 백엔드를 구현하였다. 자바 프로젝트 빌드는 그레이들*Gradle*을 이용하였다.

- 운영체제: macOS 15 Sequoia
- 도구: Visual Studio Code
- 언어 및 프레임워크
  - 프론트엔드: ReScript 11, ReScript React
  - 백엔드: Java 24, Javalin

3 Development Plan

개발 일정 이 프로젝트은 다음 일정에 따라 진행되었다.

- 프로젝트 설계: 5/19
- 프로젝트 상세 설계: 5/26
- 구현
  - 백엔드-프론트엔드 간 통신 구현 연습을 위한 토이 프로젝트 제작: 6/10
  - 백엔드-프론트엔드 통신부: 6/16
  - 프론트엔드 구현: 6/17
  - 백엔드 구현: 6/18
- 디버깅 및 기능 개선: 6/18
- 보고서 작성: 6/18

4 Code Structure

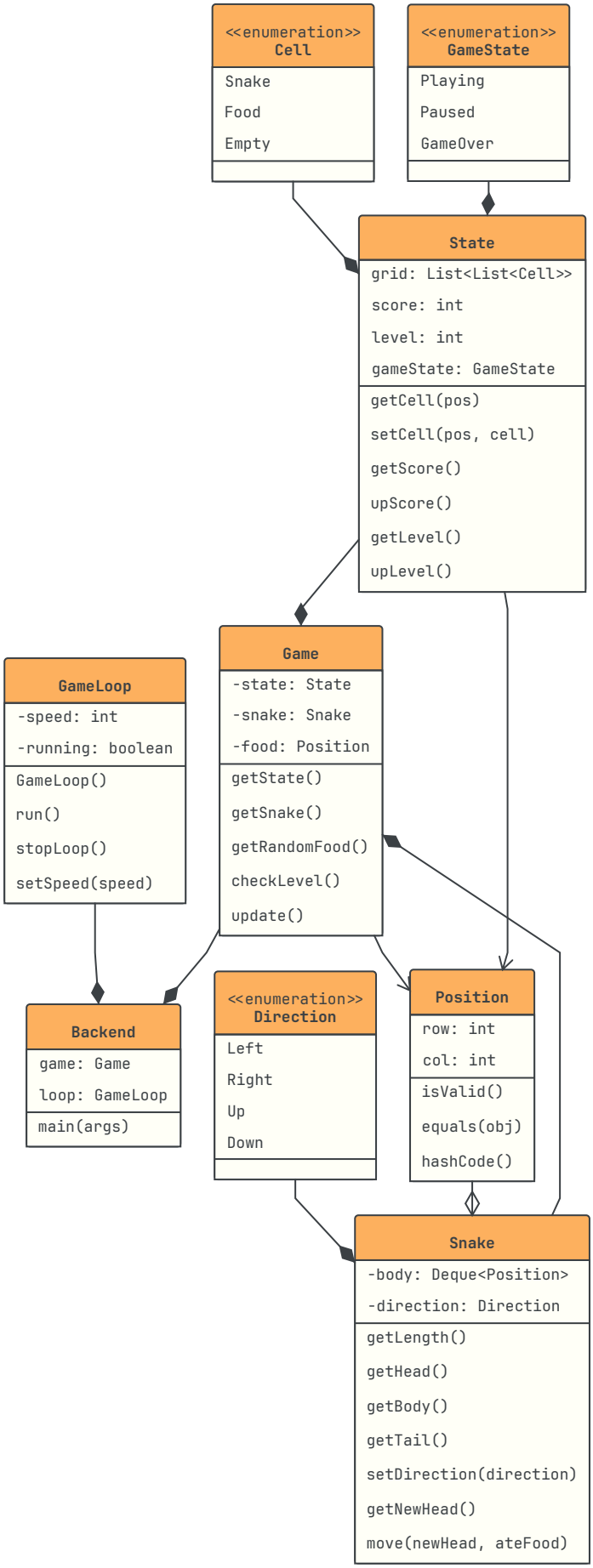
4.1 Protocol

프로토콜 SuperSnake는 `localhost:3000`에서 실행되는 프론트엔드의 요청과 `localhost:7070`에서 실행되는 백엔드의 응답으로 이루어진다. 프론트엔드 서버는 백엔드 서버의 각 엔드포인트에 요청을 보낸다. 이를테면 게임 데이터를 업데이트해야 하는 경우 `localhost:7070/game-data`에 GET 요청을 보내고, 백엔드 서버는 게임 데이터를 모아 놓은 객체 `State`를 JSON 형태로 변환하여 프론트엔드에 전달한다. 그러면 프론트엔드는 이 응답을 역직렬화하여 웹 화면에 표시하고, 뱀의 움직임 등을 업데이트한다.

쓰임에 따라 백엔드 서버의 엔드포인트를 다음과 같이 세분하였다.

- `/initialize`: 프론트엔드가 게임을 초기화했음을 백엔드에 알리는 경로이다.
- `/game-data`: 프론트엔드가 백엔드로부터 게임 데이터를 얻을 때 요청하는 경로이다.
- `/set-direction`: 프론트엔드가 받은 키보드 입력을 백엔드에 전달하는 경로이다.
- `/toggle-pause`: 게임 일시정지 상태에 변화가 일어났음을 프론트엔드가 백엔드에 알리는 경로이다.

4.2 Backend


<<enumeration>>
Cell
Snake
Food
Empty
<<enumeration>>
GameState
Playing
Paused
GameOver
State
grid: List<List<Cell>>
score: int
level: int
gameState: GameState
getCell(pos)
setCell(pos, cell)
getScore()
upScore()
getLevel()
upLevel()
GameLoop
-speed: int
-running: boolean
GameLoop()
run()
stopLoop()
setSpeed(speed)
Game
-state: State
-snake: Snake
-food: Position
getState()
getSnake()
getRandomFood()
checkLevel()
update()
Backend
game: Game
loop: GameLoop
main(args)
<<enumeration>>
Direction
Left
Right
Up
Down
Position
row: int
col: int
isValid()
equals(obj)
hashCode()
Snake
-body: Deque<Position>
-direction: Direction
getLength()
getHead()
getBody()
getTail()
setDirection(direction)
getNewHead()
move(newHead, ateFood)

5 Development progress

프로젝트의 개발 경과는 깃허브 저장소 `bloomwayz/super-snake`에서 확인할 수 있다. 이 저장소에는 프로젝트의 소스 코드와 함께, 개발 과정에서 작성한 커밋 메시지가 기록되어 있다.

5.1 Toy Project

아울러, 프로젝트의 개발 경과 가운데 주요한 것으로 연습 프로젝트를 제작한 것을 이 보고서에 싣는다. 본 프로젝트를 시작하기 전, 통신의 개념을 이해하고 리스크립트에 보다 더 익숙해지기 위해, 프론트엔드와 백엔드 사이에 문자열을 주고받는 간단한 통신을 구현해 보았다. 아래에 코드 일부를 제시한다.

5.1.1 Backend

```
// create backend server
var app = Javalin.create(config -> {
    config.bundledPlugins.enableCors(cors -> {
        cors.addRule(it -> {
            it.allowHost("http://localhost:3000");
        });
    });
}).start(7070);

// 'hello' endpoint
app.get("/hello", ctx -> {
    ctx.result("Hello, world!");
});
```

자바린을 통해 구현한 백엔드 서버 얼개이다. `Javalin.create`를 통해 서버를 생성하며, 괄호 안에 있는 명령어들은 CORS 설정을 위한 것이다. CORS란, 교차 출처 리소스 공유*Cross-Origin Resource Sharing*의 준말로 출처가 다른 서버 간의 리소스 공유를 허용할 것인지에 대한 것이다. 이 프로젝트의 경우, 프론트엔드 서버와 백엔드 서버의 출처가 다르기 때문에 CORS 설정이 필수적이며, 프론트엔드 서버가 사용하는 주소인 `http://localhost:3000`과의 공유를 허용해 두었다.

그 아래에는 `hello`라는 이름의 엔드포인트를 만들어 두었다. 프론트엔드 서버에서 `localhost:7070/hello`에 요청을 보내면, 백엔드 서버는 `Hello, world!`라는 문자열을 반환하게 된다.

5.1.2 Frontend

리스크립트와 리액트를 통해 구현한 프론트엔드 서버이다. `localhost:7070/hello`에 요청을 보내면, `Hello, world!`라는 문자열을 받게 되고, 이를 `setText`를 통해 업데이트하는 코드이다. 다음 이미지와 같이 문자열이 잘 출력되는 모습을 볼 수 있다.

```
@react.component
let make = () => {
  open React

  let (text, setText) = useState(() => "")
  let (name, setName) = useState(() => "")

  let getText = async () => {
    let response = await Fetch.get("http://localhost:7070/hello")
    let text = await response->Fetch.Response.text
    setText(_ => text)
  }

  useEffect0(() => {
    getText()->ignore
    None
  })

  <div>
  <h1> {React.string(text)} </h1>
  </div>
}
```

6 Issues & Solutions

지금 이 프로젝트에 있는 문제점은 다음과 같다.

- 디자인 문제: 격자판의 모든 칸이 회색으로만 칠해져 있어 뱀을 조작하는 데 어려움이 있다. 이는 격자판의 각 칸을 체스판처럼 교대로 채색하여 해결할 수 있다.
- 단계 및 속도 문제: 게임의 단계 및 뱀의 이동 속도를 다소 임의적으로 설정하였다. 개발 과정에서도 단계를 몇 단계로 나눌 것이고, 각 단계에서는 뱀을 어느 정도 속력으로 움직일 것이며, 어떤 성과를 달성하면 단계를 높일 것인지에 대한 고민을 계속하였으나, 시간이 부족하여 이 문제에 대해서는 깊이 생각해보지 못하고 프로젝트를 마무리하였다.

7 Program Manual

7.1 How to Run

- 1단계: 필요한 의존성이 있는 경우 설치한다. 프로젝트 빌드 도구 `gradle`과 패키지 관리자 `npm`이 필요하다.
- 2단계: 다음 명령어를 실행하여 백엔드를 실행한다.

  ```
  cd backend
  gradle run
  ```

- 3단계: 다음 명령어를 실행하여 프론트엔드를 실행한다.

  ```
  cd ../frontend
  npm install
  npm run re:build
  npm run start
  ```