Snake - implementacja w javascript

Oskar Firlej

oskar.firlej@student.put.poznan.pl

1 Utwórz podstawowe pliki

1. index.html

```
<!DOCTYPE html>
<html>

<head>
    <meta charset="UTF-8">
    <title>Snake</title>
    <script src="https://code.jquery.com/jquery-3.6.0.min.js" integrity="sha256-/xUj+3OJ
    <link rel="stylesheet" type="text/css" href="style.css">
</head>

<body>

<script type="text/javascript" src="lib.js"></script>
<script type="text/javascript" src="main.js"></script>

</body>

</html>
```

2. style.css

```
1 html, body {
2     overflow: hidden;
3     background-color: grey;
4     padding: 0%;
5     margin: 0%;
6 }
```

3. lib.js - definicje funkcji

4. main.js - główny plik z którego wywoływane będą funkcje z lib.js

2 Canvas

Wewnątrz `body` utwórz element canvas.

Rozmiar canvasu możesz ustawić na większy, lub po prostu powiększyć go na stronie (ctrl + scroll). Jeśli powiększysz go w kodzie pamiętaj aby zachować stosunek szerokości do wysokości 16:10.

```
<canvas id="canvas" width="320" height="200"></canvas>
```

W pliku `main.js` zdefiniuj dwie globalne zmienne `canvas`) i `ctx`.

Poniżej do właściwości onload obiektu window przypisz pustą funkcję anonimową.

```
let canvas, ctx;

window.onload = () => {
    //
}
```

W niej przypisz do zmiennej canvas element canvas z pliku index.html, oraz na podstawie canvasu wydobądź context i przypisz go do ctx.

Zaraz poniżej narysuj czerwony kwadrat na koordynatach x:100, y:100 i rozmiarze 50x50.

```
canvas = document.getElementById('canvas');
ctx = canvas.getContext('2d');

ctx.fillStyle = "red";
ctx.fillRect(100, 100, 50, 50);
```

3 Main loop

Poniżej window.onload zdefiniuj funkcję loop i uzywając setInterval ustaw wywoływanie jej 10 razy na sekundę.

```
window.onload = () => {
    ...
    setInterval(loop, 1000/10);
}
function loop() {
    console.log("test");
}
```

Po odświeżeniu strony można zaobserwować wielokrotnie logowany w konsoli tekst "test".

Jeśli zadziała możesz wykomentować logowanie.

4 Klasa Vector

Wewnątrz lib.js zdefiniuj klasę Vector która będzie zawierała właściwości x,y i metody add, mul, copy, oraz equals. Obiekty tej klasy będą opisywac pozycje fragmentów na które składa się wąż, oraz opisywac kierunek w którym będzie się poruszał.

Utwórz również alias `vec` dla klasy `Vector` który będzie zwracał nowy obiekt tej klasy.

```
class Vector {
    constructor(x, y) {
        this.x = x;
        this.y = y;
    }
    add(v) {
        this.x += v.x;
        this.y += v.y;
        return this;
    }
    mul(n) {
        this.x *= n;
        this.y *= n;
        return this;
    }
    copy() {
        return new Vector(this.x, this.y);
    }
    equals(v) {
        return this.x == v.x && this.y == v.y;
    }
}
let vec = (x,y) => new Vector(x,y);
```

Aby przestestowac poprawne działanie klasy `Vector` utwórz w `main.js` globalne zmienne `let pos = vec(0,0)`, oraz `dir let dir = vec(1,1);`.

```
let pos = vec(0,0);
let dir = vec(1,1);
```

Przenieś rysowanie kwadratu z funkcji `window.onload`, do `loop`, oraz zmiejsz rozmiar kwadratu do 10 pikseli. Niech kwadrat będzie rysowany w pozycji x,y zmiennej pos.

Również w funkcji loop użyj metody add na zmiennej pos żeby aktualizować zmienną pos.

```
ctx.fillStyle = "red";
ctx.fillRect(pos.x, pos.y, 10, 10);
pos.add(dir);
```

Powinna rysować się diagonalna linia z górnego lewego rogu w kierunku prawo i w dół. Dzieje się tak ponieważ nie czyścimy canvasu po zakończeniu rysowania każdej klatki.

Aby to zrobić użyj funkcji ctx.clearRect, albo ctx.fillRect żeby wyczyścić canvas na określownym obszarze, albo narysować prostakąt o określonym kolorze, który będzie rysowany jako tło przy każdej klatce.

```
ctx.clearRect(0, 0, canvas.width, canvas.height);
// or
ctx.fillStyle = "rgb(50,50,150";
ctx.fillRect(0, 0, canvas.width, canvas.height);
```

5 Input z klawiatury

Aby kontrolować poruszanie się kwadratu po ekranie dodajmy najpierw nasłuchiwanie na wciśnięcie przycisku.

```
window.addEventListener("keydown", (event) => {
    console.log(event);
    // console.log(event.key);
})
```

W konsoli można teraz zaobserwować jaką strukturę ma event przekazywany jako argument do funkcji anonimowej którą zdefiniowaliśmy jako callback po wciśnięciu przycisku na klawiaturze.

Utwórz switch case dla wartości `event.key` i aktualizuj wartość zmiennej `dir` odpowiednimi wartościami dla kazdej ze strzałek.

```
switch (event.key) {
    case "ArrowUp":
        dir = vec(0, -1);
        break;
    case "ArrowLeft":
        dir = vec(-1, 0);
        break;
    case "ArrowDown":
        dir = vec(0, 1);
        break;
    case "ArrowRight":
        dir = vec(1, 0);
        break;
    default:
        break;
}
```

Kwadrat powinien zmieniać kierunek ruchu po wciśnięciu odpowiedniego przycisku.

6 Klasa snake

Utwórz w pliku lib.js klasę Snake. Niech obecna funkcjonalność gry się nie zmienia.

Snake powinien zawierać atrybuty pos, dir, i w konstrzuktorze przyjmować obiekt pos.

Początkowa pozycja powinna być ustawiona na losową w obszarze canvasu. użyj właściwości canvas.width, oraz canvas.height.

```
// lib.js
class Snake {
    constructor(pos) {
        this.pos = pos;
        this.dir = vec(1, 0);
    }
}
```

W main.js zadeklaruj zmienną s i w funkcji window.onload przypisz do s nowy obiekt klasy Snake.

```
// main.js
let s;

window.onload = () => {
    canvas = document.getElementById('canvas');
    ...
    s = new Snake(vec(
        Math.floor(Math.random() * canvas.width),
        Math.floor(Math.random() * canvas.height)
    ));
    ...
}
```

Mimo, że wartości float w pos na razie nic by obecnie nie zepsuły to lepiej operować na liczbach całkowitych ze względu na dalsze zmiany które wprowadzimy.

Utwórz w klasie Snake dwie metody: - draw - przenieś kod z funkcji loop odpowiadający za rysowanie kwadratu do tej metody - update - przenieś aktualizowanie pozycji węża do tej funkcji

```
// lib.js
update() {
    this.pos.add(this.dir);
}

draw() {
    ctx.fillStyle = "red";
    ctx.fillRect(this.pos.x, this.pos.y, 10, 10);
}
```

```
// main.js
function loop() {
    ...
    s.update();
    s.draw();
    ...
}
```

Event listener odpowiadający za zmianę kierunku poruszania się węża również powinien odnosić się teraz do właściwości dir węża.

Usuń globalne zmienne pos i dir.

7 Obszar gry oparty na kafelkach

Utwórz w main.js stałe COLS o wartości 16 i ROWS o wartości 10, oraz zmienną TS (tile size), która będzie wyrażać w pikselach rozmiar pojedynczego kwadratu z obszaru gry.

Utworzyliśmy canvas o stosunku szerokości do wysokości 16:10. Zróbmy w takim razie również taki rozmiar planszy.

```
// main.js
const COLS = 16;
const ROWS = 10;
```

```
let TS = 320 / 16; // tile size
```

Należy wprowadzić również zmiany w losowaniu początkowej pozycji węża, oraz w sposobie jego rysowania.

```
// main.js
s = new Snake(vec(
    Math.floor(Math.random() * COLS),
    Math.floor(Math.random() * ROWS)
));

// lib.js
draw() {
    ctx.fillStyle = "red";
    ctx.fillRect(this.pos.x * TS, this.pos.y * TS, TS, TS);
}
```

Utworzymy wersję gry Snake gdzie przechodzenie przez ściany zapętlna węża na druga stronę ekranu. Należy odpowiednio modyfikować pozycję węża po zaktualizowaniu jego pozycji.

```
// lib.js

if (this.pos[0].x >= COLS) {
    this.pos[0].x = 0;
} else if (this.pos[0].y >= ROWS) {
    this.pos[0].y = 0;
} else if (this.pos[0].x < 0) {
    this.pos[0].x = COLS - 1;
} else if (this.pos[0].y < 0) {
    this.pos[0].y = ROWS - 1;
}
```

8 Wąż zamiast kwadratu

Na razie naszym wężem jest pojedynczy kwadrat. Zmodyfikujmy klasę Snake tak by pozycja węża była określona tablicą wektorów zamiast pojedynczym wektorem. Niech zerowy element określa głowę węża.

Pozycję która jest przyjmowana jako argument w konstruktorze przypisz do Właściwość pos w konstrzuktorze powinna być tablica której jedynym emelentem bedzie właśnie podana pozycja.

Początkowy kierunek ustawmy również na (0,0), żeby wąż zaczął się poruszać dopiero po wciśnięciu którejś ze strzałek.

Wszystkie odniesienia do this.pos powinny teraz odnosić się do this.pos[0].

```
// lib.js
constructor(pos) {
    this.pos = [pos];
    this.dir = vec(0, 0);
}
```

W lib.js zdefiniuj prototyp last do klasy Array by dodać metodę do tablic w JavaScript, która będzie zwracać ostatni element dowolnej tablicy. Dzięki temu nasz dalszy kod będzie bardziej czytelny.

Dodaj metodę grow do klasy snake, która doda na koniec tablicy this.pos kopię jej ostatniego elementu.

Ustawmy początkowy rozmiar węża jako np. 3. Trzeba więc w konstruktorze dwa razy wywołać metodę grow.

```
// lib.js
...
Array.prototype.last = function() {
    return this[this.length - 1];
}
class Snake {
    constructor(pos) {
        this.pos = [pos];
        this.dir = vec(1, 0);

        this.grow();
        this.grow();
    }
    grow() {
        this.pos.push(this.pos.last().copy());
    }
...
```

Wąż ma teraz długość 3, ale pozostałe części węża nie są ani aktualizowane ani rysowane. Wprowadź odpowiednie zmiany do metod draw, oraz update klasy Snake tak by pozostałe części węża podążały za głową.

Pamiętaj o tym, że przypisując do zmiennej istniejący obiekt klasy Vector odnosimy się do niego po referencji. Aby utworzyć nowy Vector użyj metodę copy, którą utworzyliśmy wcześniej.

Głowę węża możesz rysować w innym kolorze by gra była bardziej czytelna.

```
// lib.js
update() {
    for (let i = this.pos.length - 1; i > 0; i--) {
        this.pos[i] = this.pos[i - 1].copy();
    }
    this.pos[0].add(this.dir);
    ...
}
draw() {
    ctx.fillStyle = "lime";
    ctx.fillRect(this.pos[0].x * TS, this.pos[0].y * TS, TS, TS);
    ctx.fillStyle = "red";
    for (let i = 1; i < this.pos.length; i++) {
        ctx.fillRect(this.pos[i].x * TS, this.pos[i].y * TS, TS, TS);
    }
}
```

9 Jedzenie

W main.js utwórz globalną zmienną food która będzie wektorem o losowej pozycji na planszy, jej wartość zdefiniuj w window.onload.

Dodaj do klasy snake metodę eats, która będzie przyjmowała jako arguemnt vektor który bedzie jedzeniem. W tej metodzie sprawdź czy pozycja głowy węża, oraz jedzenia są sobie równa (metoda equals klasy Vector). Jeśli są to wywołaj metodę grow i zwróć true, a w przeciwnym wypadku zwróć false.

W main.js w funkcji loop po wywołaniu s.update() sprawdź czy funkcja s.eats(food) z przekazaną zmienną food zwraca true. Jeśli zwraca true to przypisz zmiennej food nowy losowy wektor, żeby wylosować nową pozycję dla jedzenia.

Dodaj również w funkcji loop rysowanie jedzenia. Najlepiej wybierz jakiś wcześniej nieużywany kolor.

```
// main.js
let food;

window.onload = () => {
    ...
    food = vec(
        Math.floor(Math.random() * COLS),
        Math.floor(Math.random() * ROWS)
    );
    ...
}

function loop() {
    ...
    s.update();

    if (s.eats(food)) {
        food = vec(
            Math.floor(Math.random() * COLS),
            Math.floor(Math.random() * ROWS)
        );
    }

    ctx.fillStyle = "rgb(250, 100, 100)";
    ctx.fillRect(food.x * TS, food.y * TS, TS, TS);
    ...
}
```

10 User interface - score

Chcielibyśmy wyświetlać gdzieś na ekranie wynik jaki obecnie mamy. Można stowrzyć zmienną score i iterowac za każdym razem jak wąż urośnie, albo odność się po prostu do długości tablicy pos węża.

Dodaj metodę score do klasy Snake, która zwraca długość tablicy pos pomniejszoną o startową długość tablicy.

```
// lib.js
score() {
    return this.pos.length - 3;
}
```

W funkcji loop w main.js narysuj po środku na górze canvasu jaki jest obecnie score.
Żeby nie definiować na szywko w pikselach rozmiar teksu można wziąć np. 10% wysokości canvasu.

```
// lib.js
// draw score
ctx.font = Math.floor(canvas.height * 0.1) + "px Arial";
ctx.fillStyle = "purple";
ctx.textAlign = "center";
ctx.fillText(s.score(), canvas.width/2, 0.1 * canvas.height);
```

11 User interface - ekrany

Dobrze by było gdyby po załadowaniu strony gra nie uruchamiała się od razu tylko najpierw pojawiłby się ekran powitalny z przyciskiem "PLAY"i dodatkowo np. na górze logo gry.

Zamiast surowego elementu <canvas> utwórz podaną poniżej strukturę w body pliku index.html. W swoim kodzie uwzględnij również wykomentowany div.

```
<div id="gameArea">
    <div id="gameScreen" class="screen">
        <canvas id="canvas" width="320" height="200"></canvas>
        <!-- <div id="highscore">0</div> -->
    </div>

    <div id="startScreen" class="screen">
        <div id="play">PLAY</div>
    </div>
</div>
```

Do pliku style.css wklej poniższy kod.

```
/* GAME AREA */

/* #gameArea {
    position: absolute;
    top: 50%;
    left: 50%;
    transform: translate(-50%, -50%);
} */

/* SCREENS */

.screen {
    display: none;
    position: absolute;
    width: 100%;
    height: 100%;
    top: 0;
    left: 0;
}
```

```
20
21 /* START SCREEN */
22
23 #play {
24     position: absolute;
25     background-color: white;
26     width: 40%;
27     height: 20%;
28     left: calc(50% - 40% / 2);
29     top: 40%;
30     font-size: 5vw;
31     display: flex;
32     align-items: center;
33     justify-content: center;
34 }
35
36 /* GAME SCREEN */
37
38 #highscore {
39     position: absolute;
40     left: 2%;
41     top: 2%;
42     font-size: 5vw;
43     color: yellowgreen;
44 }
45
46 /* canvas {
47     position: absolute;
48     width: 100%;
49     height: 100%;
50 } */
```

Na końcu body dodaj również krótki skrypt tag, a w nim zdefiniuj co ma się dziać po kliknięciu przycisku play. startScreen powinien zniknąć, a gameScreen powinien się pojawić. Możesz dodać pojawianie się gameScreen jako callback po zakończeniu chowania startScreen, lub po prostu jedno po drugim (jednocześnie jedno będzie znikać a drugie się pojawiać).

Użyj metod fadeIn i fadeOut biblioteki jQuery.

Ponieważ wszystkie screeny są defaultowo ukryte dodaj na końcu fadeIn startScreen, żeby można było rozpocząć interakcję z grą.

```
1 <script>
2   $('#play').click(() => {
3     $("#startScreen").fadeOut('fast', () => {
4             $("#gameScreen").fadeIn('fast');
5         });
6   });
7
8     $("#startScreen").fadeIn('fast');
9 </script>
```

12 Skalowanie gry na różne ekrany

Na razie wszystko działa, ale w sumie canvas jest bardzo mały i jest w lewym górnym rogu. Przeskalujemy i ustawimy canvas tak by zawsze był wyśrodkowany w pionie i w poziomie, oraz żeby wykorzystywał maksymalnie dużo ekranu zachowując przy tym zdefiniowany na poczatku stosunek szerokości do wysokości, niezależnie od tego jaki jest rozmiar ekranu.

Odkomentuj zakomentowane fragmenty kodu CSS w pliku style.css

W pliku main.js w dowolnym miejscu w pliku zdefiniuj funkcję windowResized() i zapisz ją jako funkcja do window.onresize.

```
// main.js
window.onresize = windowResized;

function windowResized() {

    let gameArea = document.getElementById('gameArea');

    let widthToHeight = COLS / ROWS;
    let newWidthToHeight = newWidth / newHeight;

    let newWidth = window.innerWidth;
    let newHeight = window.innerHeight;

    if (newWidthToHeight > widthToHeight) {
        // window width is too wide relative to desired game width
        newWidth = newHeight * widthToHeight;
    } else {
        // window height is too high relative to desired game height
        newHeight = newWidth / widthToHeight;
    }

    gameArea.style.height = newHeight + 'px';
    gameArea.style.width = newWidth + 'px';
}
```

Użyjemy tutaj przeskalowywania canvasu do rozmiaru rodzica czyli gameScreen, który z kolei dostosowuje się do rozmiaru gameArea.

Wystarczy, że w funkcji windowResized modyfikować będziemy rozmiar gameArea i canvas powinien się odpowiednio przeskalować na ekranie razem z gameArea.

13 Zapisywanie najlepszego wyniku - localStorage

Odkomentuj fragment kodu w index.html `<div id=score>0</div>`. Upewnij się, że kod CSS w style.css odnoszący się do elementu highscore jest odkomentowany. W lewym górnym rogu canvasu powinno pojawić się 0. W tym miejscu będziemy wyświetlać highscore.

Zaimplementujmy highscore. Niech bedzie to właściwość węża. Dodaj w konstruktorze Snake zmienną highscore i zainicjuj jej wartośc na 0.

W metodzie grow po zwiększeniu rozmiaru węża dodaj sprawdzenie czy score jest większy niż highscore. Jeśli jest zaktualizuj wartość highscore do obecnego score, oraz zaktualizuj zawartość elementu highscore.

```
grow() {
    this.pos.push(this.pos.last().copy());
```

```
3
4       if (this.score() > this.highscore) {
5           this.highscore = this.score();
6           $("#score").html(this.score());
7       }
8   }
```

Mimo, że nie ma obecnie zaimplementowanego mechanizmu końca gry, to wiadomo że wartości zmiennych w skryptach js są tracone po odświeżeniu strony. Do gry wchodzi tutaj localStorage, czyli pamięć przeglądarki, którą można wykorzystać do zapisywania danych pomiędzy sesjami.

Dużo stron internetowych wykorzystuje właśnie localStorage do przechowywania preferencji użytkownika do różnych funkcjonalności na stronie. Np. poziom głośności na youtube jest zapamiętywany po odświeżeniu strony.

Zanim wprowadzimy jakiekolwiek zmiany sprawdź jaka jest zawartość zmiennej localStorage. Wpisz localStorage w konsoli przegladarki. Zwrócony powinien zostać obiekt klasy Storage.

Dodajmy teraz zapisywanie highscore do localStorage. Uzyj metody setItem. Jako klucz użyj stringa "highscoreą jako wartość highscore, które jest właściwością węża. Zapisywanie tej wartości do localStorage powinno się odbywać zaraz po wykryciu nowego lepszego wyniku, czyli w tym samym miejscu gdzie aktualizujemy zawartość highscore.

```
1 localStorage.setItem("highscore", this.highscore);
```

Odśwież stronę. Zbierz co najmniej jedno jedzenie i sprawdź ponownie zawartość localStorage w konsoli. Powinna pojawićsię w obiekcie jedna wartość o kluczu highscore. Odświez ponownie i zanim zjesz jakieś jedzenie upewnij się że wartośc zapisana w localStorage ciągle się tam znajduje.

Teraz można dodać odczytywanie localstorage w konstruktorze Snake. Sprawdź czy w localStorage znajduje się jakaś wartość poprzez przyrównanie zwróconej wrtości do null. Jeśli nie ma przypisz wartość score i zaktualizuj zawartość elementu highscore.

Pamiętaj by umieścić ten kod po podwójnym wywołaniu metody grow() w konstruktorze, ponieważ zanim wąz urośnie podwójnie w konstruktorze metoda score() zwróciłaby wartość -2.

```
1 this.highscore = localStorage.getItem('highscore') === null ? this.score() : Number
    (localStorage.getItem('highscore'));
2 $("#highscore").html(this.highscore);
```

Mamy teraz dwa elementy tekstowe. Liczba punktów - rysowana na canvasie, oraz highscore - liczba jako tekst w html. Tekst rysowany na canvasie nie jest wydajną metodą rysowania UI. Dużo tekstu przy dużych rozdzielczościach może bardzo obciążać procesor. Dodatkowo przy mniejszych rozdzielczościach mogą być widoczne artefakty.

14 Rysowanie obrazów

Dodamy teraz grafikę do canvasu. Najpierw musimy załadować obrazy. Użyjemy grafiki w postaci `sprite`, czyli jeden plik .png zawierający wiele różnych grafik w różnych miejscach. Zależnie od tego którego fragmentu oryginalneg pliku będziemy chcieli użyć tego użyjemy poprzez podanie odpowiedniej lokalizacji w png.

Utwórz dwie tablice asocjacyjne (dict). Pierwszą nazwij sources i do elementu śnake"przypisz link do obrazu. Drugą nazwij sources i zainicjuj ją jako pustą.

```
let sources = {
    sprite: 'https://rembound.com/files/creating-a-snake-game-tutorial-with-html5/snake-graphics.png'
}
let images = {}

let loadedImages = 0;
for (src in sources) {
    images[src] = new Image();
    images[src].onload = function () {
        loadedImages++;
        if (loadedImages == Object.keys(sources).length) {
            imagesLoaded();
        }
    };
    images[src].src = sources[src];
}

function imagesLoaded() {
    console.info("Images loaded!");
    $("#startScreen").fadeIn('slow');
}
```

Każde ze zdjęć jest ładowane i w funkcji onload każdego pojedynczego zdjęcia licznik wczytanych zdjęć jest porównywany ze źródłową ilością zdjęć. Dopiero jeśli ilość wszystkich wczytanych zgadza się z ilością źródeł podanych to funkcja imagesLoaded jest uruchamiana. Dopiero wtedy ekran startowy się pojawia i można rozpocząć gre.

Dzięki temu unikamy pokazywania użytkownikowi gry z jeszcze niewczytanymi grafikami.

Dla przykładu będziemy rysować głowę węża. W pliku png zlokalizuj na których pikselach znajduje się grafika głowy węża (obojętnie której), oraz na jak duże kwadraty jest podzielony sprite.

Zastąp kwadrat rysowany w innym kolorze jako głowę węża wykorzystując do tego fragment załadowanego obrazu.

```
const S_TS = 64;
ctx.drawImage(
    images.sprite, // image
    S_TS*4, S_TS*1, S_TS, S_TS, // position in image
    this.pos[0].x * TS,this.pos[0].y * TS, TS, TS // position on canvas
);
```

Głowa węża powinna być teraz rysowana, ale jeszcze nie jest odpowiednio zrotowana.

Obraz głowy węża będziemy rotować zależnie od właściwości dir klasy Snake. Pierwsze co trzeba zrobićto zapisać kontekst.

- Zapisz konteks.
- uzyj funkcji translate by przetłumaczyć kontekst na środek komórki w której będziemy rysować głowę węża.
- Zrotuj kontekst. Uzyj funkcji Math.atan2() i przekaż do niej wartości x i y zmiennej dir, by uzyskać informację o kierunku w formie radianów.

- Zmodyfikuj obecną formę rysowania głowy tak by była rysowana w odpowiedniej lokalizacji (pamiętaj, że przetłumaczyliśmy wcześniej kontekst).
- Przywróć kontekst.

```
const S_TS = 64; // sprite tilesize
ctx.save();
ctx.translate(this.pos[0].x * TS + TS/2,this.pos[0].y * TS + TS/2);
ctx.rotate(-Math.atan2(this.dir.x, this.dir.y));
ctx.drawImage(
    images.sprite,
    S_TS*4, S_TS*1, S_TS, S_TS,
    -TS/2,  -TS/2, TS, TS);
ctx.restore();
```

15 Zadania

15.1 Kolizje

Dodaj mechanizm przegranej do gry. Przy update węża sprawdź czy pozycja głowy nie jest taka sama jak którejś z części węża. Zresetuj wtedy grę.

15.2 Sterowanie

Obecnie sterowanie nie działa do końca poprawnie. Dodaj ograniczenie by nie dało się zręcać o 180 w jednym ruchu. podpowiedź: nowy kierunek możesz porównywać z odwrotnością obecnego kierunku węża.

Opcjonalne: Upewnij się, że zaimplementowany mechanizm ograniczający sterowanie rzeczywiście je ogranicza. Tzn. W przypadku gdy wąż jest skierowany w prawo szybko (w jednej klatce gry) klikniesz góra i później lewo to wąż nie skręci o 180 stopni.

15.3 Dodatkowe grafiki

Dodaj grafikę rysowania jedzenia, oraz ogonu (ostatniego fragmentu) węża.