

Projekt - Gra Gracz

Obiekt gracza

Gracz będzie opisany szeregiem zmiennych odpowiedzialnych głównie za jego wygląd i ruch. Do tych najważniejszych, implementowanych na początku należą:

- position - lokalizacja gracza na planszy poziomu określana punktem środkowym. W przykładzie poniżej gracz znajduje się na środku pola [1, 1]. **Każdy student powinien wybrać swoje miejsce startowe dla obiektu gracza.**
- r - promień koła reprezentującego obiekt gracza. Promień jest wartością mniejszą niż połowa wysokości ściany, aby gracz mógł przejść przez najniższe korytarze o wysokości pojedynczej ściany.
- colour - kolor koła reprezentującego obiekt gracza. **Każdy student powinien wpisać swój własny kolor.**

```
game.js
var position = {
    x: level.wall.width * (1 + 0.5),
    y: level.wall.height * (1 + 0.5)
};
var r = (level.wall.height / 2) - 1;
var colour = '#0000ff';
```

Znając podstawowe parametry obiektu gracza, należy stworzyć nową klasę Player z konstruktorem przyjmującym dwie wartości: kolor gracza i poziom, po którym gracz ma się poruszać. Konstruktor ustawia tym samym trzy powyższe zmienne. To jest zadanie do samodzielnego wykonania przez studenta.

Rysowanie gracza

Za wizualizację gracza będzie odpowiedzialna nowa metoda draw(context) w klasie Player. Najpierw ustawiany jest kolor figury, a następnie rysowany jest łuk będący tak naprawdę kołem. To jest zadanie do samodzielnego wykonania przez studenta.

Więcej na temat rysowania koła w obiekcie canvas można przeczytać na stronie https://www.w3schools.com/tags/canvas_arc.asp.

Następnie należy stworzyć gracza i wywołać jego rysowanie na aktualnym poziomie w grze.

```
game.js
player = new Player('#0000ff', level);
player.draw(context);
```

Poruszanie się gracza

Żeby gracz mógł poruszać się, musi mieć jakąś prędkość pionową i poziomą. W związku z tym należy dodać nowy parametr *velocity* w konstruktorze klasy *Player*. Przyjmuje on na razie wartości zerowe, ponieważ bohater w grze nie porusza się bez interwencji gracza.

```
game.js
this.velocity = {
    x: 0,
    y: 0
};
```

Wartość nowego parametru jest rozumiana tak naprawdę jako odcinek drogi pokonywany w danym odstępie czasu, jakim jest czas pomiędzy odświeżeniami funkcji. Jest on dodawany do obecnej pozycji gracza w metodzie *move (context)*. Na koniec wywoływane jest rysowanie gracza, aby zwizualizować jego nową pozycję na mapie gry.

```
game.js
move(context) {
    this.position = {
        x: this.position.x + this.velocity.x,
        y: this.position.y + this.velocity.y
    }
    this.draw(context);
}
```

Gracz może poruszać się w lewo (klawisz strzałki w lewo) i w prawo (klawisz strzałki w prawo) oraz może skakać (klawisz strzałki w góre). Gdy gracz naciśnie konkretny klawisz odpowiedzialny za ruch bohatera w grze, wartość parametru *velocity* jest ustawiana. Dzieje się to w metodzie *startMoving (key, context)*.

```
game.js
startMoving(key, context) {
    switch(key) {
        case 37:
            this.velocity = {x: -1, y: this.velocity.y};
            break;
        case 38:
            this.velocity = {x: this.velocity.x, y: -1};
            break;
        case 39:
            this.velocity = {x: 1, y: this.velocity.y};
            break;
    };
    this.move(context);
}
```

Który z powyższych kodów jest odpowiednikiem klawisza strzałki w lewo, w prawo i w góre?

Więcej na temat kodów klawiszy w JavaScript można przeczytać na stronie
https://www.w3schools.com/jsref/event_key_code.asp.

Wywołanie metody *startMoving (key, context)* następuje po wykryciu zdarzenia wcisnięcia klawisza. Odpowiedzialny za to *EventListener* jest dodany po definicji instancji gracza.

```
game.js
document.addEventListener('keydown', function(event) {
    player.startMoving(event.keyCode, context);
});
```

W podobny sposób należy dodać reakcję na puszczenie klawisza. Wykrycie zdarzenia keyup powinno uruchomić nową metodę stopMoving() w klasie Player, która to zeruje wartości parametru velocity u gracza. To jest zadanie do samodzielnego wykonania przez studenta.

Odświeżanie gry

Aktualny kod pozwala nam na poruszanie się w grze, ale bohater zostawia za sobą ślad. Aby poprzednie pozycje gracza zniknęły, potrzebne jest cykliczne uruchamianie funkcji wizualizującej wszystkie figury elementu canvas. Funkcja drawGame() czyści zawartość płótna i wywołuje metody rysowania poziomu gry oraz gracza. Jest ona uruchamiana 25 razy w ciągu 1 sekundy, aby ruch gracza był płynny dla ludzkiego oka.

```
game.js
function drawGame() {
    context.clearRect(0, 0, canvas.width, canvas.height);
    level.drawMap(context);
    player.move(context);
}

setInterval(drawGame, 1000/25);
```

Dlaczego najpierw rysowany jest poziom gry, a potem dopiero gracz? Czy kolejność wywoływania funkcji rysujących jest ważna?

Więcej na temat funkcji „czasowych” w JavaScript można przeczytać na stronie https://www.w3schools.com/js/js_timing.asp.

Prędkość pozioma i pionowa

W aktualnej wersji kodu prędkość pozioma i pionowa przyjmuje wartość równą 1 w funkcji startMoving(key, context). Może jednak istnieć potrzeba, żeby ruch poziomy był szybszy lub wolniejszy, niż ruch pionowy, czyli skok. Ponadto, należy usystematyzować tą wartość i uzależnić ją od rozmiaru ścian poziomu gry, aby przy innej mapie ruch gracza był poprawnie skonfigurowany. W związku z tym, należy dodać kolejny parametr (speed) do konstruktora klasy Player. Wartości tego parametru mogą być zmienione przez każdego studenta.

```
game.js
this.speed = {
    x: level.wall.width / 5,
    y: level.wall.height / 4.5
};
```

Mając zdefiniowany parametr speed, należy go zastosować w odpowiednim miejscu w metodzie startMoving(key). To jest zadanie do samodzielnego wykonania przez studenta.

Naprawa ruchu gracza

Aktualny ruch gracza może doprowadzić do konfliktu pomiędzy jednocześnie wciśniętymi przyciskami klawiatury. W efekcie gracz nie będzie się poruszał, mimo, że ma wciśnięty poprawny klawisz ruchu. Aby tego uniknąć wprowadzony jest kolejny parametr `moving` w konstruktorze klasy `Player`. Informuje on, które klawisze ruchu są wciśnięte, czyli jaki ruch jest aktywny. Domyślnie gracz nie porusza się.

```
game.js
this.moving = {
    left: false,
    right: false,
    up: false
}
```

Metody `startMoving(key)` i `stopMoving(key)` zostają uaktualnione w taki sposób, żeby ustawały poprawnie parametr pomocniczy `moving`. Poniżej pokazany jest kod dla klawisza lewej strzałki. Analogicznie należy wykonać to dla klawisza prawej i górnej strzałki. To jest zadanie do samodzielnego wykonania przez studenta.

```
game.js
startMoving(key) {
    switch(key) {
        case 37:
            this.moving.left = true;
            break;
        [...]
    };
    this.setMoving();
}

stopMoving(key) {
    switch(key) {
        case 37:
            this.moving.left = false;
            break;
        [...]
    };
    this.setMoving();
}
```

W obu metodach rozpoczęjących i kończących ruch gracza wywoływana jest nowa metoda `setMoving()` odpowiedzialna za wyznaczenie aktualnej prędkości gracza, która to może być sumą kilku składowych. Przykładowo, wciskając lewą i prawą strzałkę, gracz powinien pozostać w miejscu, bo prędkość pozioma gracza będzie sumą dodatniej i ujemnej szybkości, czyli będzie równa 0. Poniżej pokazany jest kod dla aktywności klawisza lewej strzałki. Analogicznie należy wykonać to dla klawisza prawej i górnej strzałki. To jest zadanie do samodzielnego wykonania przez studenta.

```
game.js
setMoving() {
    this.velocity = {x: 0, y: 0};
    if(this.moving.left) this.velocity.x -= this.speed;
    [...]
}
```

Grawitacja

Żeby gracz nie poruszał się tak dowolnie, prawie jak w próżni, potrzebna jest grawitacja. Jest to cecha danego poziomu, dlatego w klasie Level powinniśmy dodać kolejny prywatny parametr gravity oraz adekwatne dla niego metody get i set. Wartość grawitacji najlepiej jest uzależnić w pewien sposób od wysokości ściany. To jest zadanie do samodzielnego wykonania przez studenta.

```
game.js
this.#gravity = wallHeight / 25;
```

Następnie w konstruktorze klasy Player należy dodać dwa parametry:

- gravity - zmienna ułatwiająca późniejsze używanie wartości grawitacji zdefiniowanej na danym poziomie.
- fallingCounter - zmienna pomocnicza odwzorowująca w pewien sposób działanie grawitacji na ruch gracza.

```
game.js
this.gravity = level.gravity;
this.fallingCounter = 1;
```

Grawitacja działa tylko na pionowy ruch gracza. Zgodnie z prawami fizyki, z każdym odświeżeniem metody move(context), prędkość pionowa gracza powinna maleć aż do 0. Ponieważ prędkość pionowa gracza velocity jest niezmienna, jego przemieszczanie się w pionie korygowane jest przez parametr fallingCounter. Aby grawitacja faktycznie spowalniała gracza i sprowadziła go ostatecznie na „ziemię”, czyli na ścianę typu solid, konieczne jest zwiększenie zmiennej fallingCounter w każdym ponownym wywołaniu metody.

```
game.js
move(context) {
    this.position = {
        x: this.position.x + this.velocity.x,
        y: this.position.y + this.velocity.y + this.gravity * this.fallingCounter
    }
    this.draw(context);
    this.fallingCounter += 0.15;
}
```

Wartość parametru fallingCounter jest cały czas powiększana, zatem trzeba przywrócić go do stanu początkowego w niektórych sytuacjach. Przykładem jest sam początek skakania gracza, czyli moment, kiedy w metodzie startMoving(key) zmienna moving.up jest jeszcze równa false.

```
game.js
if(!this.moving.up) this.fallingCounter = 1;
```

Inną sytuacją jest rezygnacja gracza ze skakania, czyli moment, kiedy puszczasz klawisz strzałki w góre. W metodzie stopMoving(key) należy wtedy przywrócić wartość początkową dla parametru fallingCounter. To jest zadanie do samodzielnego wykonania przez studenta.

Ściany dookoła gracza

Aby gracz mógł reagować na środowisko, w którym się znajduje, potrzebna jest znajomość ścian, z którymi ma styczność. W tym celu należy wyznaczyć indeksy położenia kolejnych punktów skrajnych dla obiektu reprezentującego gracza, czyli dla koła. Inaczej mówiąc, w metodzie move (context) należy wyliczyć, na której ścianie znajduje się aktualnie dana część gracza:

- centerX, centerY - punkt środkowy koła, czyli aktualna pozycja gracza podzielona przez szerokość lub wysokość ściany poziomu. Użyta jest funkcja podłoga Math.floor, aby zamienić wyliczoną liczbę zmienoprzecinkową na odpowiednią liczbę całkowitą.
- left - skrajny punkt koła po lewej stronie. Od aktualnej pozycji gracza należy odjąć promień koła, ponieważ w ten sposób zdefiniowany jest układ współrzędnych obiektu canvas. Dodatkowo odejmowana jest wartość 0.01, która koryguje problematyczne sytuacje przy końcowym zaokrąglaniu wartości w dół.
- right - skrajny punkt koła po prawej stronie, który należy wyliczyć analogicznie do definicji powyższych punktów. To jest zadanie do samodzielnego wykonania przez studenta.
- top - skrajny punkt koła na górze, który należy wyliczyć analogicznie do definicji powyższych punktów. To jest zadanie do samodzielnego wykonania przez studenta.
- bottom - skrajny punkt koła na dole, który należy wyliczyć analogicznie do definicji powyższych punktów. To jest zadanie do samodzielnego wykonania przez studenta.

Dlaczego przy końcowej zamianie liczby zmienoprzecinkowej na całkowitą używana jest funkcja podłoga, a nie funkcja sufit lub zaokrąglania?

```
game.js
var location = {
  centerX: Math.floor(this.position.x / this.level.wall.width),
  centerY: Math.floor(this.position.y / this.level.wall.height),
  left: Math.floor((this.position.x - this.r - 0.01) / this.level.wall.width),
  right: [...],
  top: [...],
  bottom: [...]
};
```

Następnie należy znaleźć konkretne ściany, z którymi mają styczność punkty skrajne gracza. Znając indeksy położenia koła, należy pobrać ścianę:

- center - ściana, na której znajduje się środek gracza.
- left - ściana, na której znajduje się lewy skrajny punkt gracza.
- right - ściana, na której znajduje się prawy skrajny punkt gracza. To jest zadanie do samodzielnego wykonania przez studenta na podstawie powyższych punktów.
- top - ściana, na której znajduje się górny skrajny punkt gracza. To jest zadanie do samodzielnego wykonania przez studenta na podstawie powyższych punktów.
- bottom - ściana, na której znajduje się dolny skrajny punkt gracza. To jest zadanie do samodzielnego wykonania przez studenta na podstawie powyższych punktów.

```
game.js
var walls = {
  center: this.level.walls[ this.level.map[location.centerY][location.centerX] ],
  left: this.level.walls[ this.level.map[location.centerY][location.left] ],
  right: [...],
  top: [...],
  bottom: [...]
};
```

Reakcja na ściany solid

Gracz powinien zatrzymać się, jeżeli dotknie ściany typu solid. Jednakże, w tym samym czasie każdy jego ruch w inną stronę powinien być dozwolony. Rozważając przypadek ściany lewej, należy sprawdzić dwa warunki:

1. Czy ściana lewa jest typu solid.
2. Czy gracz chce poruszać się w lewą stronę, czyli czy chce wejść w ścianę typu solid.
W tym celu porównywane są dwie pozycje gracza: poprzednia zapisana w nowej zmiennej `oldPosition` oraz nowa przechowywana jako parametr `position`.

Jeżeli oba warunki są spełnione, pozioma pozycja gracza jest ustawiana na wartość maksymalną w stosunku do ściany lewej. Gracz zatrzymuje się dokładnie przy tej ścianie.

Analogicznie należy wykonać sprawdzenie dla ściany prawej, górnej i dolnej. To jest zadanie do samodzielnego wykonania przez studenta.

```
game.js
var oldPosition = this.position;

this.position = {
    x: this.position.x + this.velocity.x,
    y: this.position.y + this.velocity.y + this.gravity * this.fallingCounter
};
this.fallingCounter += 0.15;

if(walls.left.solid == 1) {
    if(this.position.x < oldPosition.x)
        this.position.x = (location.left + 1) * this.level.wall.width + this.r;
}
[...]

this.draw(context);
```

Dodatkowo, po opadnięciu, czyli po zetknięciu się dolnej skrajnej części gracza ze ścianą typu solid, siła grawitacji powinna wrócić do początkowej wartości.

```
game.js
this.fallingCounter = 1;
```

Teraz należy tylko uporządkować kod, aby był bardziej czytelny. W tym celu należy przenieść 4 funkcje warunkowe do jednej nowej metody `checkSolidWalls(walls, location, oldPosition)` zdefiniowanej w klasie `Player` i wywołać ją w odpowiednim miejscu. To jest zadanie do samodzielnego wykonania przez studenta.

Wygrana

Metoda odpowiedzialna za sprawdzanie wygranej gracza została dodana do klasy `Player` jako `checkWinWalls(walls)`. W funkcji sprawdzane jest, czy któraś skrajna część gracza dotyka ściany z id równym `win` i jeżeli tak, wyświetlany jest odpowiedni komunikat, a gracz jest przenoszony na początkową pozycję dla danego poziomu. W ten sposób poziom restartuje się.

```

checkWinWalls(walls) {
    if( walls.top.id =='win' || walls.bottom.id == 'win' ||
        walls.left.id =='win' || walls.right.id == 'win') {

        alert(' WIN! :) ');
        this.position = {
            x: this.level.wall.width * (1 + 0.5),
            y: this.level.wall.height * (1 + 0.5)
        };
    }
}

```

Przegrana

Analogicznie do metody sprawdzającej wygraną, należy dodać metodę sprawdzającą przegraną gracza - checkLoseWalls(walls). To jest zadanie do samodzielnego wykonania przez studenta.

Wskakiwanie w ściany ukośne w stosunku do położenia gracza

Wskakiwanie w ściany ukośne jest jednym z pojawiających się problemów w tym rozwiążaniu. Demonstruje go film `problem1_ścianyUkośne.mp4`. Spowodowany on może być brakiem sprawdzania ścian położonych ukośnie w stosunku do położenia gracza. Zadaniem każdego studenta jest naprawienie tego błędu.

Skakanie w locie

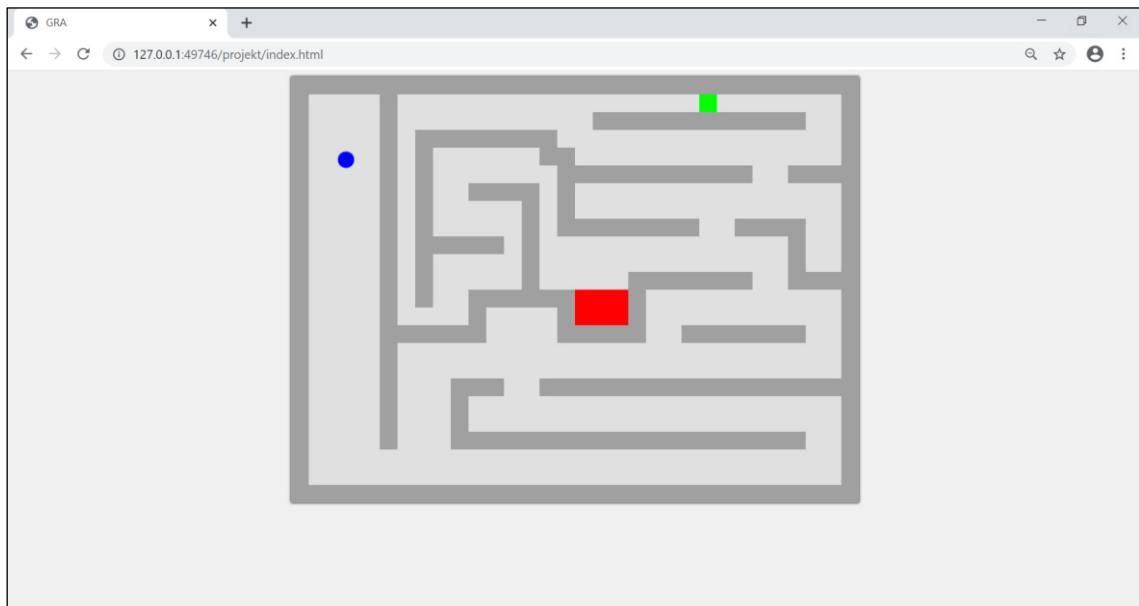
W obecnej sytuacji gracz może podskakiwać w powietrzu i w ten sposób wspinać się. Problem pokazany jest na filmie `problem2_skakanieWLocie.mp4`. Kolejnym krokiem jest ograniczenie tej możliwości do skoku tylko w sytuacji, kiedy gracz stoi na ścianie typu solid. To jest zadanie do samodzielnego wykonania przez studenta.

Wiszenie pod sufitem

Kiedy gracz jest w trakcie skoku i doryka sufitem, jego pozycja wertykalna pozostaje bez zmian. Problem zademonstrowany jest na filmie `problem3_wiszenie.mp4`. Zgodnie z logiką po zderzeniu się z sufitem, gracz powinien zacząć opadać. To jest zadanie do samodzielnego wykonania przez studenta.

Efekt końcowy

Po ukończeniu tej części, projekt powinien wyświetlać zdefiniowaną mapę z 4 rodzajami ścian oraz gracza, którym można się poruszać. Ponadto, gracz może wygrać po wejściu na zielony obszar lub umrzeć po dotknięciu obszaru czerwonego. Poniżej Rysunek 1 jako przykład.



Rysunek 1. Efekt końcowy treści przedstawionej powyżej.