



WYDZIAŁ
ELEKTROTECHNIKI
I INFORMATYKI
POLITECHNIKI RZESZOWSKIEJ



RAG-2

Rzeszów University of Technology Games
for Artificial Intelligence 2.0

Tworzenie minigier w standardzie RUT-AI Games 2.0

Marcin Bator
173592

Paweł Buczek
173599

Bartłomiej Krówka
173650

Opiekun: **dr inż. Dawid Kalandyk**

Tworzenie nowych gier

Wstęp

Zaimplementowane w aplikacji gry są napisane w języku TypeScript. Logika wspólna dla wszystkich gier została wydzielona do klasy `BaseGameComponent`, co pozwala programiście tworzącemu nową grę na skupienie się wyłącznie na logice specyficznej dla gry i ściąga z niego obowiązek dostosowywania logiki komunikacji z innymi mikroserwisami i modelami AI. Do stworzenia strony wizualnej gry można użyć elementu HTML `canvas`, który pozwala na dynamiczne renderowanie obiektów 2D. Jako przykładu stworzenia nowej gry posłuży gra `flappybird`.

Wygenerowanie plików gry

Na początku należy wybrać nazwę gry. Nazwa nie powinna zawierać spacji, pauz, podkreślników - jedynie małe litery alfabetu angielskiego, a także powinna być jak najkrótsza.

W głównym katalogu projektu `rag-2-games` uruchomić komendę, która uruchomi skrypt tworzący pliki niezbędne do utworzenia gry:

```
npm run games:new flappybird
```

Pliki w których powinna znaleźć się implementacja gry zostaną przez skrypt umieszczone w folderze `projects/rag-2-games-lib/src/lib/games/flappybird/`. W tym katalogu znajdują się pliki `flappybird.component.ts` oraz `models/flappybird.class.ts`. Po uruchomieniu skryptu dostępna stanie się także podstrona o adresie URL `/game/flappybird`.

Definicja gry

W pliku `flappybird.class.ts` powinny znaleźć się dwie klasy: `FlappyBirdState` dziedzicząca po `TGameState` oraz `FlappyBird` dziedzicząca po `Game`.

Klasa `FlappyBirdState` powinna zawierać właściwości specyficzne dla gry, np. pozycję gracza, prędkość, punkty, itp. Wartości te będą przesyłane do serwisu AI i zapisywane w bazie danych. Należy używać ich podczas implementacji gry. Przykładowa klasa znajduje się na listingu 1.

Listing 1: Klasa `FlappyBirdState`

```
1 export class FlappyBirdState implements TGameState {
2   public birdY = 0;
3   public birdSpeedY = 0;
4   public gravity = 0;
5   public jumpPowerY = 0;
6   public obstacleSpeed = 0;
7   public score = 0;
8   public difficulty = 0;
9   public obstacles = Array.from({ length: 4 }, () => ({
10     distanceX: 0,
11     centerGapY: 0,
12   }));
13   public isGameStarted = false;
14 }
```

Klasa FlappyBird powinna nadpisywać pola klasy Game, po której dziedziczy. Należy ustawić nazwę gry, zmienną stanu z użyciem klasy FlappyBirdState, zmienną tekstową ze specyfikacją danych z klasy stanu oraz tablicę graczy. Przykładowa klasa znajduje się na listingu 2.

Listing 2: Klasa FlappyBird

```
1 export class FlappyBird extends Game {
2   public override name = 'flappybird';
3   public override state = new FlappyBirdState();
4   public override outputSpec = "
5     output:
6     birdY: float, <0, 600>;
7     birdSpeedY: float, <-inf, inf>;
8     gravity: float, <0.5, 1>;
9     jumpPowerY: float, <5, 15>;
10    obstacleSpeed = <2, 10>;
11    score: int, <0, inf>;
12    difficulty: int, <0, inf>;
13    obstacles: [{distanceX: int, <0, 1900>, centerGapyY: int <100,
14      500>}];
15    isGameStarted: boolean;
16
17    default values:
18    birdY: 300;
19    birdSpeedY: 0;
20    gravity: 0.5;
21    jumpPowerY: 10;
22    obstacleSpeed: 2;
23    score: 0;
24    difficulty: 1;
25    isGameStarted: false;
26  ";
27  public override players = [
28    new Player(
29      0,
30      true,
31      'Player 1',
32      { jump: 0 },
33      {
34        ': {
35          variableName: 'jump',
36          pressedValue: 1,
37          releasedValue: 0,
38        },
39      },
40      '<jump>: value of {0, 1}, 0: not jump, 1: jump',
41      { jump: '[SPACE]', start: '[SPACE]' }
42    ),
43  ];
44 }
```

Tablica `players` jest kluczowym elementem działania rozgrywki i połączenia z modelami zewnętrznymi. Każdy element tablicy to obiekt klasy `Player`, który zawiera informacje o graczu, jego akcjach, opisie akcji, przypisanych klawiszach oraz opisie klawiszy. Pola wymagane przez konstruktor klasy `Player` zostały przedstawiona na listingu 3.

Listing 3: Klasa `Player`

```
1 export class Player {
2   public id: number;
3   public isObligatory: boolean;
4   public name: string;
5   public inputData: TExchangeData = {};
6   public controlsBinding: Record<string, IPlayerControlsBinding> = {};
7   public expectedDataDescription = '';
8   public controlsDescription: TExchangeData = {};
9   public playerType: PlayerSourceType;
10 }
```

Pole `isObligatory` określa, czy gracz jest obowiązkowy, czy też może być nieaktywny (np. w grze Chińczyk może być od 2 do 4 graczy, więc 2 z nich jest obowiązkowa, a 2 nie).

Pole `inputData` powinno być obiektem typu klucz-wartość, gdzie kluczem jest nazwa parametru wejściowego gracza, zaś wartością: wartość domyślna parametru. W przypadku gry `FlappyBird` gracz ma tylko jeden parametr wejściowy: ruch. W sterowaniu klawiaturą wartość parametru jest zmieniana przez spację i jest równa 1, gdy gracz naciska spację, a 0, gdy nie naciska. Dostęp do wartości parametru odbywa się poprzez pobranie `inputData['jump']` z odpowiedniego elementu tablicy `players`.

Zachowanie klawiatury jest opisane w polu `controlsBinding`. To również obiekt typu klucz-wartość, gdzie kluczem jest symbol klawisza (w tym przypadku ' ', czyli spacja), zaś wartością obiekt typu `IPlayerControlsBinding`. Zawiera on 3 pola: pole `variableName` określa nazwę parametru wejściowego (zdefiniowanego wyżej), pole `pressedValue` określa wartość parametru, gdy klawisz jest wciśnięty, a pole `releasedValue` określa wartość parametru, gdy klawisz jest zwolniony. Dzięki temu podczas implementacji gry nie trzeba martwić się o obsługę klawiatury, ponadto logika zostanie też prawidłowo obsłużona przez serwis AI.

Ostatnie dwa pola zawierają jedynie informacje dla użytkownika. Pierwsze z nich opisuje, jakie wartości mogą przyjmować parametry wejściowe gry. Jest to istotne z punktu widzenia programisty implementującego serwis integrujący się z grą. Drugie z pól służy do wyświetlania, jakie klawisze są przypisane do akcji.

Implementacja gry

Skrypt generujący komponent stworzył w nim element `canvas`, a także licznik FPS i deklaracje metod potrzebnych do zapisania logiki gry. Przykładowy dekorator klasy komponentu z logiką gry znajduje się na listingu 4.

Listing 4: Dekorator komponentu gry

```
1 @Component({
2   selector: 'app-flappybird',
3   standalone: true,
4   imports: [CanvasComponent],
5   template: `<div>
6     score: <b>{{ game.state.score }}</b>
7     >, difficulty: <b>{{ game.state.difficulty }}</b>
8     >, jumpPower: <b>{{ game.state.jumpPowerY }}</b>
9     >, gravity: <b>{{ game.state.gravity }}</b>
10    >, obstacle speed: <b>{{ game.state.obstacleSpeed }}</b>
```

```

11     ><b>
12         {{
13             game.state.jumpPowerY === 15 &&
14             game.state.gravity === 1 &&
15             game.state.obstacleSpeed === 10
16             ? ', MAXIMUM DIFFICULTY HAS BEEN REACHED!'
17             : ''
18         }}
19     </b>
20 </div>
21 <app-canvas #gameCanvas></app-canvas> <b>FPS: {{ fps }}</b> ',
22 })

```

Dekorator zawiera ponadto wyświetlanie różnych wartości stanu gry powyżej okienka canvas. Pod nim wyświetla się wartość klatek na sekundę.

Klasa obowiązkowo powinna posiadać pole `game` nadpisujące pole z klasy nadrzędnej, będące typu `FlappyBird`, a więc klasy utworzonej w poprzednim kroku. Rzutowanie typu należy wykonać w metodzie `ngOnInit`, wywołując uprzednio metodę z klasy nadrzędnej.

W klasie mogą znaleźć się też pola niezwracane do websocketu, które zawierają informacje nieistotne dla działania gry, takie jak szerokość gracza lub przeszkody. Jeśli klasa będzie nadpisywać metody `OnDestroy` lub inne z klasy nadrzędnej, należy odpowiednio uzupełnić nagłówki klasy. Należy przy tym przestrzegać zasad dziedziczenia zawartych w dokumentacji klasy `BaseGameComponent`, np. obowiązkowe wywołanie metody klasy nadrzędnej w metodzie przeładowanej.

Najczęściej w klasie gry powinny zostać przeładowane metody `ngAfterViewInit`, w której zostanie wywołany reset stanu gry i pierwsze wyrenderowanie zawartości, metoda `restart` restartująca stan gry oraz metoda `update` aktualizująca stan gry. We wszystkich tych metodach należy pamiętać o wywołaniu metody z klasy nadrzędnej.

Pozostała część kodu powinna renderować stan gry na elemencie canvas i odpowiednio obsługiwać wydarzenia używając właściwości stanu zdefiniowanych w klasie gry. **Nie powinno się implementować własnego przechwytywania zdarzeń klawiatury, ponieważ nie będą one obsługiwać poprawnie komunikacji przez websocket.** Przykładowa implementacja głównych metod klasy i wywołanie pozostałych znajduje się na listingu 5.

Listing 5: Klasa `FlappyBirdComponent`

```

1 export class FlappyBirdComponent
2 extends BaseGameWindowComponent
3 implements OnInit, AfterViewInit, OnDestroy
4 {
5     private _birdWidth = 20;
6     private _birdHeight = 20;
7     private _obstacleWidth = 50;
8     private _obstacleGapHeight = 200;
9     private _minDistanceBetweenObstacles = 200;
10    private _passedObstacles: boolean[] = new Array(4).fill(false);
11
12    public override game!: FlappyBird;
13
14    public override ngOnInit(): void {
15        super.ngOnInit();
16
17        this.game = this.game as FlappyBird;
18    }
19
20    public override ngAfterViewInit(): void {
21        super.ngAfterViewInit();

```

```

22         this.resetBirdAndObstacle();
23         this.render();
24     }
25
26
27     public override restart(): void {
28         this.game.state = new FlappyBirdState();
29
30         this.game.state.isGameStarted = false;
31         this.resetBirdAndObstacle();
32         this.resetScoreAndDifficulty();
33     }
34
35     protected override update(): void {
36         super.update();
37
38         if (
39             !this.game.state.isGameStarted &&
40             this.game.players[0].inputData['jump'] === 1
41         ) {
42             this.game.state.isGameStarted = true;
43         }
44
45         if (!this.isPaused && this.game.state.isGameStarted) {
46             this.updateBirdPosition();
47             this.updateObstaclePosition();
48             this.updateBirdSpeed();
49             this.checkCollision();
50         }
51         this.render();
52     }
53 }

```

Podsumowanie

Podczas tworzenia i po zaimplementowaniu gry należy ją testować, uruchamiając serwer deweloperski, który zawiera niezbędne elementy głównego projektu frontendowego. Umożliwi to korzystanie z menu bocznych, a także konsoli bez uruchamiania dodatkowych serwisów. W przypadku błędów należy sprawdzić, czy wszystkie metody zostały nadpisane, czy wszystkie zmienne są poprawnie zadeklarowane, a także czy nie ma błędów w logice gry. Można wzorować się na już zaimplementowanych grach.