
	Uniwersytet Technologiczno-Przyrodniczy im. J. i J. Śniadeckich w Bydgoszczy Wydział Telekomunikacji, Informatyki i Elektrotechniki Instytut Telekomunikacji i Informatyki	
Przedmiot	Programowania aplikacji mobilnych	
Prowadzący	dr inż.. Mirosław Miciak	
Temat	Gra River Raid	
Student	Roman Volchuk, Mateusz Kalksztejn	

Celem projektu jest napisanie gry na platformę mobilną na podstawie wybranego z listy przykładu gry na komputery 8 bitowe oraz wytworzenie dokumentacji.

I. Ogólny opis gry - koncepcja

a) Cel projektu (gry):

Głównym celem projektu jest odwozrowanie gry River Raid stworzonej przez firmę Activision na konsolę Atari 2600, a później na Atari 5200, 8-bitowe Atari, ColecoVision, Commodore 64, IBM PC, IBM PCjr, Intellivision, ZX Spectrum i MSX.

b) Użytkownicy, postaci, przedmioty:

Rozgrywkę mamy że oglądany z lotu ptaka, gracz leci myśliwcem nad rzeką bez powrotu w rajdzie za liniami wroga. Odrzutowiec gracza może poruszać się tylko w lewo i w prawo - nie może manewrować w górę iw dół ekranu - ale może przyspieszać i zwalniać. Odrzutowiec gracza rozbija się, jeśli zderzy się z brzegiem rzeki lub statkiem wroga lub jeśli skończy się paliwo. Zakładając, że paliwo można uzupełnić, a jeśli gracz uniknie obrażeń, rozgrywka jest w zasadzie nieograniczona.

Gracz zdobywa punkty za zestrzelenie wrogich **tankowców** (30 punktów), **helikopterów** (60 punktów), **składów paliw** (80 punktów), **odrzutowców** (100 punktów) i **mostów** (500 punktów). Niszczenie mostów służy również jako punkty kontrolne w grze. Jeśli gracz rozbije samolot, rozpocznie następny odrzutowiec na ostatnim zniszczonym moście.

c) Granice systemu gry (obszar, sterowanie itp.)

- Poruszanie odrzutowcem w lewo / w prawo,
- Przyspiesz na joysticku,
- Zwolnij - zwolnij na joysticku,
- Przycisk ognia,
- Wznów grę za pomocą rezerwowego dżojstika lub przycisku.



d) Lista możliwości (funkcji gry)

Użytkownik/Gracz:

- Uruchamia rozgrywkę,
- Wyłącza aplikację,
- Porusza samolocikiem,
- Strzelanie rakietami,
- Może zostać zniszczony po zderzeniu z brzegiem rzeki lub statkiem wroga lub jeśli skończy się paliwo,
- Zbieranie punktów.

Przeciwniki:

Nazwa przeciwnika	Punkty za zabicie,uwagi
Pancernik	30pkt.
Helikopter	60pkt.
Baza paliw	80pkt.,Odrzutowiec tankuje,gdy przelatuje na skadem paliwa
Jet	100pkt.
Most	500pkt.,Rzadki obiekt premiujący długą rozgrywkę



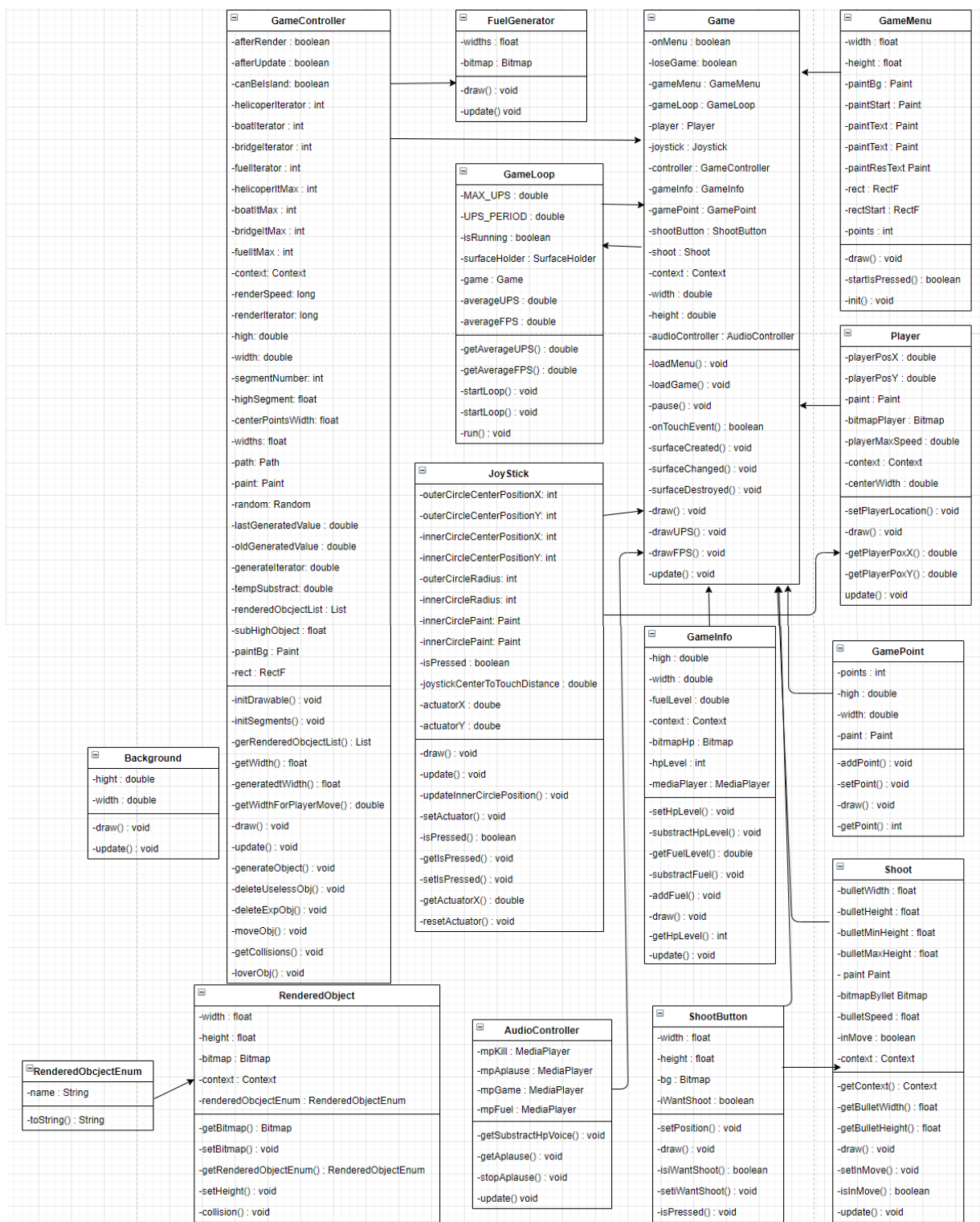
II. ANALIZA DZIEDZINY

a) Klasy zidentyfikowane w dziedzinie

Nazwa klasy	Za co odpowiada
AudioController	Wsystko co stosuje się dźwięku
Background	Rysuje tło
FuelGenerator	Odpowiada za paliwo
GameController	Główny do rysowania przedmiotów
GameInfo	Informacja
GameLoop	Rysowania wątku
GameMenu	Menu
Game	Umieszczenia wszystkich elementów w tej klasie
GamePoint	Punkty
Joystick	Sterowanie joystickiem
Player	Sterowanie graczem
RenderedObject	Renderuje objecty(helikopter,paliwo,most,tódz
Shoot	Strzały
ShootButton	Przycisk, żeby strzelać



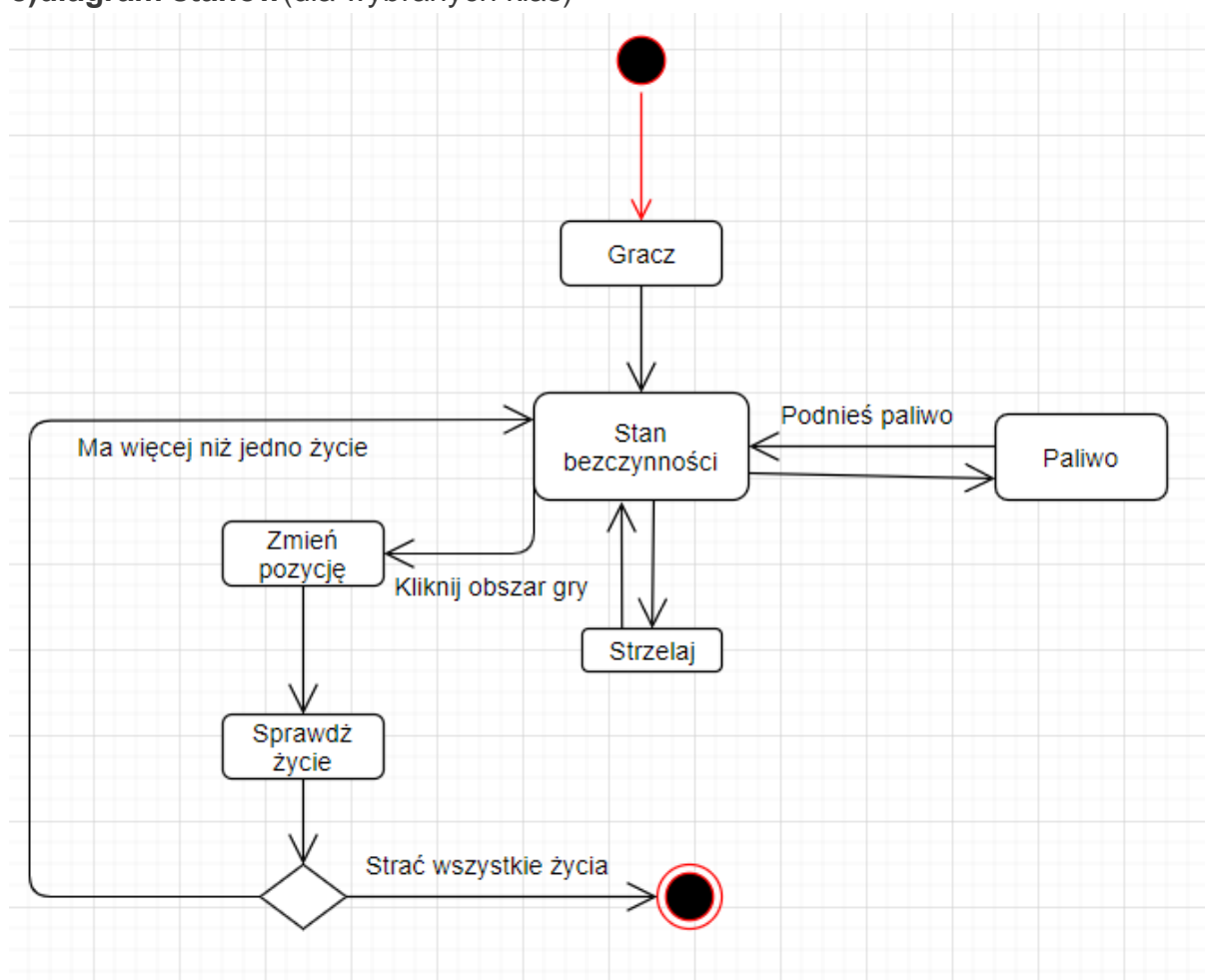
b) diagram klas



Rysunek 1 : Diagram Klas



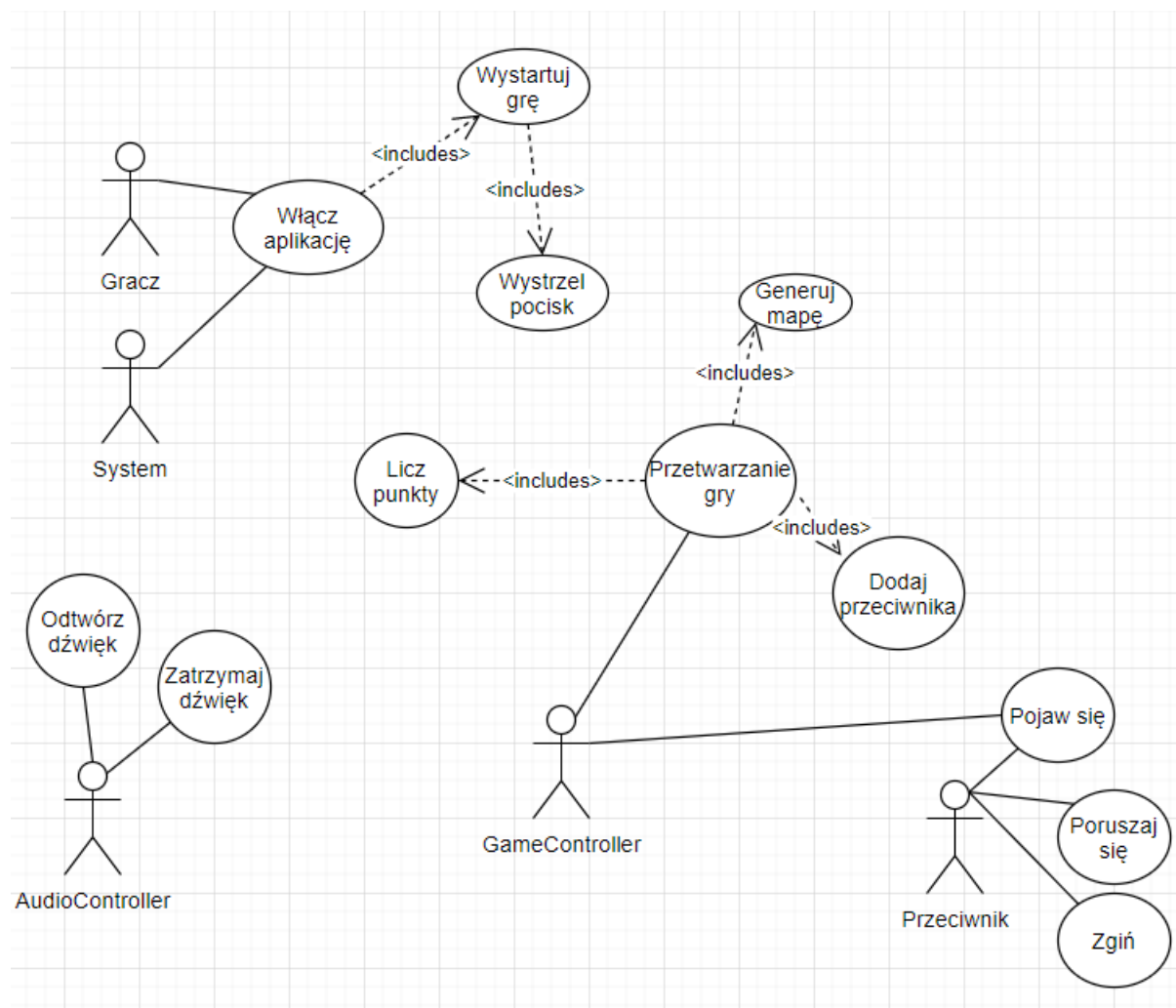
c)diagram stanów(dla wybranych klas)



Rysunek 2:Maszyna stanów dla klasy gracza

III. Specyfikacja wymagań

a) Ogólny diagram przypadków użycia



Rysunek 3: Diagram przypadków użycia



b) Definicje przypadków użycia

Sekcja	Treść
Aktorzy	Gracz, System, GameController
Warunki wstępne	Pojawienie się przeciwnika na planszy
Warunki końcowe	Zmiana pozycji samolotu
Resultat	Ruch w jednym z dwóch kierunków na osi X
Scenariusz główny	1a. Gracz posiada trzy życia i gra się nie skończyła 1b. Gracz nie posiada trzech życia i gra się skończyła 2a. Gracz klika na dwa możliwe przyciski do przemieszczania 3a. Paletka przemieszcza się w wybranym kierunku osi X

Sekcja	Treść
Aktorzy	Gracz, System, GameController, AudioController
Warunki wstępne	Gracz zdobywa paliwo i umożliwia mu to dalszą podróż
Warunki końcowe	Paliwo zwiększa dalszą odległość
Resultat	Gracz może dalej latać
Scenariusz główny	1a. Gracz zdobywa paliwo 2a. Gracz posiada już paliwo 3a. Gracz nie posiada już paliwa 4a. Odtwarzanie dźwięku 5a. Dłaska podróż

Sekcja	Treść
Aktorzy	Gracz, System, GameController Czasu, Przeciwnik
Warunki wstępne	Pojawienie się nowego przeciwnika na planszy
Warunki końcowe	Zajęcie pozycji w losowym miejscu mapy
Resultat	Poruszanie się przeciwnika w lewo lub prawo na mapie
Scenariusz główny	1a. Pojawienie się przeciwnika 2a. Przemieszczanie się po mapie 3a. Zniszczenie przeciwnika 4a. Dodanie punktów dla gracza



IV. Analiza i projekt

1. Architektura systemu gry

Na wykorzystaną przez nas architekturę gry składają się warstwy:

a) Model

Jest to zbiór klas przechowujących dane.

- Background
- FuelGenerator
- GameInfo
- GameMenu
- GamePoint
- Joystick
- RenderedObjectEnum
- ShootButton
- EnumDir

b) Menadżer

Jest to zbiór klas zapewniających uogólnione funkcjonalności, które pozwalają na łączenie mechanik pochodzących z innych klas

- Game
- GameLoop
- Player
- RenderedObject
- Shoot

c) Kontroler

Są to pojedyncze klasy udostępniające mechanikę obiektom

- AudioController
- GameController



2. Obiektowy model analizy

Nazwa	AudioController
Atrybuty	-mpAplause -mpGame -mpFuel -mpKill
Metody	-getSubstractHpVoice () -getAplause () -stopAplause () -update()

Nazwa	Background
Atrybuty	-high -width -paint -rect
Metody	-draw () -update ();

Nazwa	FuelGenerator
Atrybuty	-context -widths -bitmap
Metody	-draw () -update ();



Nazwa	Game
Atrybuty	<ul style="list-style-type: none">-onMenu-loseGame-gameMenu-gameLoop-player-joystick-controller-gameInfo-gamePoint-shootButton-shoot-context-width-height-audioController
Metody	<ul style="list-style-type: none">-loadMenu ()-loadGame ()-pause()-onTouchEvent()-surfaceCreated()-surfaceChanged()-surfaceDestroyed()-draw()-drawUPS()-drawFPS()-update()



Nazwa	GameController
Atrybuty	<ul style="list-style-type: none">-afterRender-afterUpdate-canBelsland-helicopterIterator-boatIterator-bridgeIterator-fuelIterator-helicopterItMax-boatItMax-bridgeItMax-fuelItMax-context-renderSpeed-renderIterator-high-width-segmentNumber-highSegment-centerPointWidth-widths-path-paint-random-lastGeneratedValue-oldGeneratedValue-generateIterator-tempSubstract-renderedObjectList-subHighObject-paintBg-rect
Metody	<ul style="list-style-type: none">-initDrawable ()-initSegments ()-getRenderedObjectList()-getWidths()-generateWidth()-getWidthForPlayerMove()-draw()-update()-generateObject()-deleteUselessObj()-deleteExpObj()-moveObj()-getCollisions()-loverObj()



Nazwa	GameInfo
Atrybuty	-high -width -fuelLevel -context -bitmapHp -hpLevel -mediaPlayer
Metody	-setHpLevel () -subtractHpLevel () -getFuelLevel() -subtractFuel() -addFuel() -draw() -getHpLevel() -update()

Nazwa	GameLoop
Atrybuty	-MAX_UPS -UPS_PERIOD -isRunning -surfaceHolder -game -averageUPS -averageFPS
Metody	-getAverageUPS () -getAverageFPS () -startLoop() -stopLoop() -run()

Nazwa	GameMenu
Atrybuty	-height -width -paintBg -paintStart -paintText -paintResText -rect -rectStart -points
Metody	-draw () -startIsPressed () -init()



Nazwa	GamePoint
Atrybuty	-high -width -points -paint
Metody	-addPoint () -setPoints () -setPoints() -getPoints()

Nazwa	Joystick
Atrybuty	-outerCircleCenterPositionX -outerCircleCenterPositionY -innerCircleCenterPositionX -innerCircleCenterPositionY -outerCircleRadius -innerCircleRadius -innerCirclePaint -outerCirclePaint -isPressed -joystickCenterToTouchDistance -actuatorX -actuatorX
Metody	-draw () -update () -updateInnerCirclePosition() -setActuator() -isPressed() -getIsPressed() -setIsPressed() -getActuatorX() -resetActuator()

Nazwa	Player
Atrybuty	-playerPosX -playerPosY -paint -bitmapPlayer -playerMaxSpeed -context -centerWidth
Metody	-setPlayerRotation () -draw () -getPlayerPosX() -getPlayerPosY() -update()

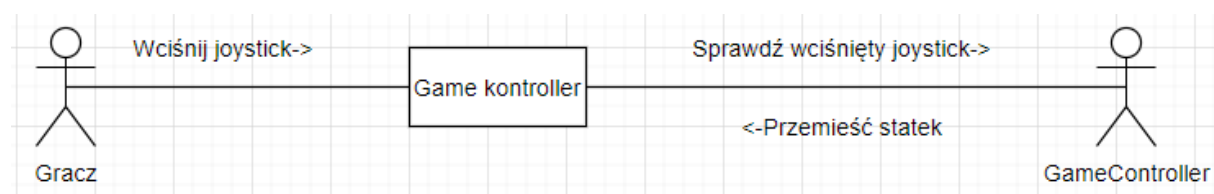
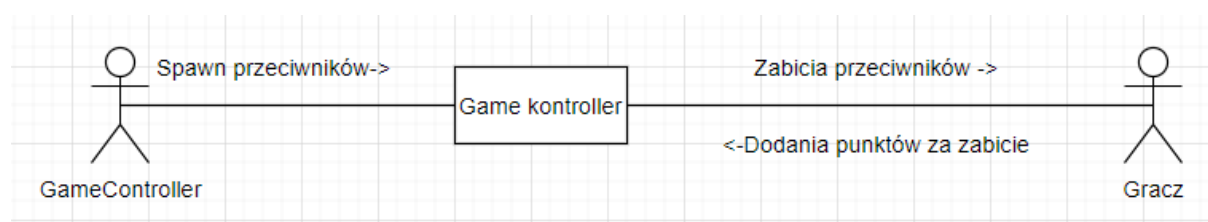
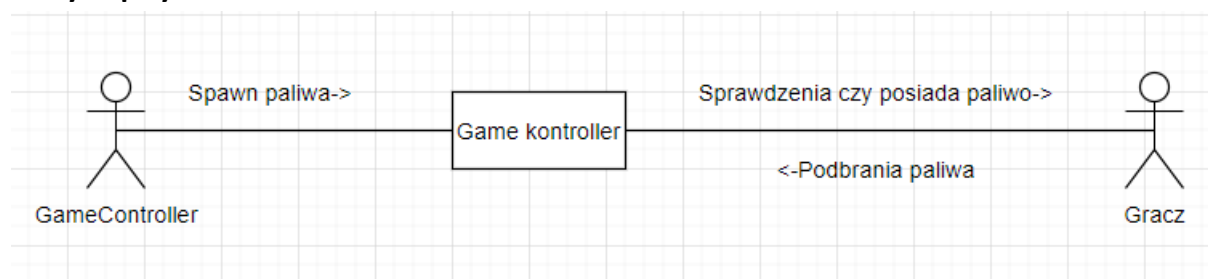


Nazwa	RenderedObject
Atrybuty	-heigh -width -bitmap -context -renderedObjectEnum
Metody	-getBitmap () -setBitmap () -getRenderedObjectEnum() -setHeight() -collision()

Nazwa	Shoot
Atrybuty	-bulletWidth -bulletHeight -bulletMinHeight -bulletMaxHeight -paint -bitmapBullet -bulletSpeed -context -inMove
Metody	-getContext () -getBulletWidth () -getBulletHeight () -draw () -setInMove () -isInMove () -update ()

Nazwa	ShootButton
Atrybuty	-heigh -width -bg -iWantShoot
Metody	-setPosition () -draw () -isiWantShoot() -setIWantShoot() -isPressed()

Klasy i opisy



3. Projekt interfejsu użytkownika IRS



Na interfejs w menu głównym składają się trzy napisy:

- UPS – limit frame per second.
- FPS – frame per second.
- Start – po kliknięciu na niego rozpoczyna się gra.
- Score – po zakończeniu gry ilość punktów

Na interfejs podczas rozgrywki składają się :

- UPS – limit frame per second.
- FPS – frame per second.
- Przycisk rakiet – pozwala na wystrzelenia rakiety
- Joystick – pozwala na poruszanie się
- Ikony statków – pozostała liczba żyć
- Score – zdobyta liczba punktów
- Ikona paliwa – pozostała liczba paliwa



V. Implementacja

```
public class MainActivity extends Activity {
    private Game game;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        Window window= getWindow();
        window.setFlags(
            WindowManager.LayoutParams.FLAG_FULLSCREEN,
            WindowManager.LayoutParams.FLAG_FULLSCREEN
        );
        game = new Game(this);
        setContentView(new Game(this));
    }
    @Override
    protected void onPause() {
        game.pause();
        super.onPause();
    }
}
```

MainActivity jest klasą uruchomieniową całej gry. Przekazywany jest w niej kontekst dla obiektu klasy **Game**.

```
public class GameLoop extends Thread {
    //ustawienie UPS i FPS
    private static final double MAX_UPS = 30;
    private static final double UPS_PERIOD = 1E+3 / MAX_UPS;
    boolean isRunning = false;
    private SurfaceHolder surfaceHolder;
    private Game game;
    private double averageUPS;
    private double averageFPS;
}
```

GameLoop klasą rozszerzającą wątek w której obsługane zostało rysowanie gry przy użyciu obiektu **canvas** oraz częstotliwość odświeżania.



```
@Override
public void run() {
    super.run();

    int updateCount = 0;
    int frameCount = 0;

    long startTime;
    long elapsedTime;
    long sleepTime;

    Canvas canvas = null;
    startTime = System.currentTimeMillis();
    while (isRunning) {
        try {
            canvas = surfaceHolder.lockCanvas();
            synchronized (surfaceHolder) {
                game.update();
                updateCount++;
                game.draw(canvas);
            }
        } catch (IllegalArgumentException e) {
            e.printStackTrace();
        } finally {
            if (canvas != null) {
                try {
                    surfaceHolder.unlockCanvasAndPost(canvas);
                    frameCount++;
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        }

        elapsedTime = System.currentTimeMillis() - startTime;
        sleepTime = (long) (updateCount * UPS_PERIOD - elapsedTime);
        if (sleepTime > 0) {
            try {
                sleep(sleepTime);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }

        while (sleepTime < 0 && updateCount < MAX_UPS - 1) {
            game.update();
            updateCount++;
            elapsedTime = System.currentTimeMillis() - startTime;
            sleepTime = (long) (updateCount * UPS_PERIOD - elapsedTime);
        }

        elapsedTime = System.currentTimeMillis() - startTime;
        if (elapsedTime >= 1000) {
            averageUPS = updateCount / (1E-3 * elapsedTime);
            averageFPS = frameCount / (1E-3 * elapsedTime);
            updateCount = 0;
            frameCount = 0;
            startTime = System.currentTimeMillis();
        }
    }
}
```

By zaimplementować wyżej przedstawione działanie należało nadpisać funkcję **run()**.



```
public class Game extends SurfaceView implements SurfaceHolder.Callback {
    private boolean onMenu;
    private boolean loseGame;
    private GameMenu gameMenu;
    private GameLoop gameLoop;
    private Player player;
    private Joystick joystick;
    private GameController controller;//todo
    private GameInfo gameInfo;
    private GamePoint gamePoint;
    private ShootButton shootButton;
    private Shoot shoot;
    private Context context;
    double width;
    double height;
    private AudioController audioController;

    public Game(Context context) {/////}
    public void loadMenu(){/////}

    public void loadGame(){/////}
    public void pause() {gameLoop.stopLoop();}
    @Override
    public boolean onTouchEvent(MotionEvent event){/////}
    @Override
    public void surfaceCreated(@NonNull SurfaceHolder holder) {
        gameLoop.startLoop();
    }

    @Override
    public void draw(Canvas canvas) {
        super.draw(canvas);
        if(onMenu) {
            gameMenu.draw(canvas);
        }else {//gra
            controller.draw(canvas);
            gameInfo.draw(canvas);
            joystick.draw(canvas);
            shootButton.draw(canvas);
            player.draw(canvas);
            shoot.draw(canvas);
            gamePoint.draw(canvas);
        }

        //FPS i UPS
        drawFPS(canvas);
        drawUPS(canvas);
    }

    public void drawUPS(Canvas canvas){
        String averageUPS = Double.toString(gameLoop.getAverageUPS());
        Paint paint =new Paint();
        paint.setColor(Color.RED);
        paint.setTextSize(50);
        canvas.drawText("UPS:"+averageUPS.substring(0,2),50,100,paint);
    }
    public void drawFPS(Canvas canvas){

        String averageFPS = Double.toString(gameLoop.getAverageFPS());
        Paint paint =new Paint();
        paint.setColor(Color.RED);
        paint.setTextSize(50);
        canvas.drawText("FPS:"+averageFPS.substring(0,2),50,200,paint);
    }

    public void update() {
        if(onMenu) {
            ///////////////
        }else {//gra
            audioController.update(gameInfo.getFuelLevel());
            //update
            if(gameInfo.getHpLevel()==0){//brak żyć wraca do start
                loseGame=true;
                onMenu=true;
                loadMenu();
            }
            if(gameInfo.getFuelLevel()<=0){//brak paliwa
                loseGame=true;
                onMenu=true;
                loadMenu();
            }
        }
    }
}
```

Klasa Game opiera się na 3 funkcjach .Są to : **update()** , **draw()** , **onTouchEvent()**

Pozostałe funkcję w klasie służą do zwiększenia czytelności kodu.



```
@Override
public boolean onTouchEvent(MotionEvent event){
    if(onMenu){ //menu
        switch (event.getAction()) {
            case MotionEvent.ACTION_DOWN:{
                if(gameMenu.startIsPressed(event.getX(), event.getY())) {
                    onMenu = false;
                    loadGame();
                }
                return true;}
        }
    }
    else { //gra
        switch (event.getAction()) {
            case MotionEvent.ACTION_DOWN:
                //shoot
                shootButton.isPressed(event.getX(), event.getY(), shoot.isInMove());

                if (joystick.isPressed(event.getX(), event.getY())) {
                    joystick.setIsPressed(true);
                }
                return true;
            case MotionEvent.ACTION_MOVE:
                if (joystick.getIsPressed()) {
                    joystick.setActuator(event.getX(), event.getY());
                }
                return true;
            case MotionEvent.ACTION_UP:
                joystick.setIsPressed(false);
                joystick.resetActuator();
                return true;
        }
    }
    return super.onTouchEvent(event);
}
```

Funkcja **onTouchEvent** odpowiada za analizowanie interakcji użytkownika z programem.

Funkcja obsługuje takie elementy jak:

- 1) Przycisk startu gry
- 2) Przycisk strzału
- 3) Joystick odpowiedzialny za sterowanie obiektem gracza.



```
@Override
public void draw(Canvas canvas) {
    super.draw(canvas);

    if(onMenu) {
        gameMenu.draw(canvas);
    }else { //gra
        controller.draw(canvas);
        gameInfo.draw(canvas);
        joystick.draw(canvas);
        shootButton.draw(canvas);
        player.draw(canvas);
        shoot.draw(canvas);
        gamePoint.draw(canvas);
    }

    //FPS i UPS
    drawFPS(canvas);
    drawUPS(canvas);
}
```

W funkcji **draw()** wykonujemy rysowanie przy użyciu obiektu canvas .

Rysowanie jest rozdzielone na kilka obiektów by zachować obiektowość projektu.

```
public void update() {
    if(onMenu) {
        ///////////////////////////////////////////////////
    }else { //gra
        audioController.update(gameInfo.getFuelLevel());

        controller.update(shoot, gamePoint, player, gameInfo, audioController, joystick);
        gamePoint.update();
        joystick.update();
        player.update(joystick, controller.getWidthForPlayerMove());
        shoot.update(shootButton, (float) player.getPlayerPosX());
        gameInfo.update();
        if(gameInfo.getHpLevel()==0){ //brak życia wraca do start
            loseGame=true;
            onMenu=true;
            loadMenu();
        }
        if(gameInfo.getFuelLevel()<=0){ //brak paliwa
            loseGame=true;
            onMenu=true;
            loadMenu();
        }
    }
}
```

Funkcja **update()** ma za zadanie dokonywać zmian w parametrach rysowanego widoku.

Występuje rozdzielenie na kilka połączonych z sobą obiektów .



```
@RequiresApi(api = Build.VERSION_CODES.N)
public void draw(final Canvas canvas) {
    if (afterUpdate) {
        afterRender = false;
        //bg render
        canvas.drawRect(this.rect, this.paintBg);

        //river render
        for (int widthInx = 0; widthInx < widths.length - 1; widthInx++) {

            this.path = new Path();
            path.moveTo(centerPointWidth - widths[widthInx], (float) (high - (highSegment * widthInx)));//1
            path.lineTo(centerPointWidth - widths[widthInx + 1], (float) high - (highSegment * (widthInx + 1)));//2
            path.lineTo(centerPointWidth + widths[widthInx + 1], (float) high - (highSegment * (widthInx + 1)));//3
            path.lineTo(centerPointWidth + widths[widthInx], (float) (high - (highSegment * widthInx)));//4
            path.lineTo(centerPointWidth - widths[widthInx], (float) (high - (highSegment * widthInx)));//5
            canvas.drawPath(this.path, this.paint);
        }
    }
}
```

W klasie **GameController** występuje implementacja rysowania mapy.

Opiera się ona na rysowaniu określonej ilości segmentów (trapezów) które mają symulować wygląd rzeki.



```
public void generateObject() {
    deleteExpObj();

    //renderowanie obiektów jest 2 razy losowane , raz obiekt a potem szerokość
    int generatedIer = random.nextInt(200);

    if((widths[widths.length-1]<=width/10)&&(bridgeIterator>=bridgeIMax)){
        renderedObjectList.add(new RenderedObject(context, RenderedObjectEnum.bridge, centerPointWidth- 330, 0));
        bridgeIterator=0;
    }

    if (renderedObjectList.size() <= 16) {
        if (generatedIer > 0 && generatedIer <= 12) {
            float objectWidth = centerPointWidth - widths[widths.length - 1] + 100 + (2 * (widths[widths.length - 1] - 100)) *
random.nextFloat();
            renderedObjectList.add(new RenderedObject(context, RenderedObjectEnum.boat, objectWidth, 0));
        }
        if (generatedIer > 12 && generatedIer <= 20) {
            float objectWidth = centerPointWidth - widths[widths.length - 1] + 100 + (2 * (widths[widths.length - 1] - 100)) *
random.nextFloat();
            renderedObjectList.add(new RenderedObject(context, RenderedObjectEnum.fuel, objectWidth, 0));
        }
        if (generatedIer > 20 && generatedIer < 30) {
            float objectWidth = centerPointWidth - widths[widths.length - 1] + 100 + (2 * (widths[widths.length - 1] - 100)) *
random.nextFloat();
            renderedObjectList.add(new RenderedObject(context, RenderedObjectEnum.helicopterP, objectWidth, 0));
        }
        ///////
        if (generatedIer > 30 && generatedIer < 40) {
            renderedObjectList.add(new RenderedObject(context, RenderedObjectEnum.shipL, (float) width, 0));
        }
        if (generatedIer > 40 && generatedIer < 50) {
            renderedObjectList.add(new RenderedObject(context, RenderedObjectEnum.shipP, 0, 0));
        }
    }
}
```

Generowanie przeciwników na mapie jest losowe i odpowiada za nie funkcja **generateObject()**



VI. Testy aplikacji

a) Błędy związane z ograniczeniami narzucanymi przez określony OS

Jedynym problemem, jaki byliśmy w stanie spostrzec, jest czas rozpoczęcia gry. Po wciśnięciu przycisku start należy odczekać kilka sekund, aż załaduje się ekran rozgrywki i udostępnione zostaną narzędzia pozwalające na interakcję z aplikacją

b) Testy na emulatorach i testy na realnych urządzeniach

Gra była testowana zarówno na emulatorach jak i dwóch osobnych telefonach o różnych specyfikacjach. W obu tych przypadkach działa prawidłowo. Rozgrywka jest płynna i w trakcie przechodzenia przez kolejne poziomy nie występują problemy z grafiką, mechaniką czy też oprawą audio. Jedynym większym problemem jest wcześniej wymieniona w pierwszym podpunkcie, długość przejścia pomiędzy ekranem menu głównego, a ekranem rozgrywki.

c) Testy poprawności instalacji i deinstalacji na różnych urządzeniach/emulatorach.

Aplikacja instaluje się bezproblemowo na urządzeniach z systemem android. Testy przeprowadzane były od wersji systemu 6.0 „Marshmallow”, aż do najnowszej wersji znajdującej się w emulatorze NOX. Dezinstalacja aplikacji przebiegła również bez żadnych komplikacji zarówno w wypadku emulatora jak i dwóch testowanych telefonów.

d) Testy związane z ilością wykorzystywanej pamięci

Do testów ilości zużywanych zasobów wykorzystany został telefon Huawei MYA-L41 z systemem Android 6.0 (2GB RAM, 4 rdzenie 1.4 GHz, rozdzielczość 720x1280) oraz dwie aplikacje (jedna to stworzona na potrzeby tego projektu gra „River Raid”, a druga dostępna do pobrania na Google Play gra „Milionerzy”). Poniżej znajduje się porównanie w postaci zrzutów ekranu z telefonu obrazujących wykorzystanie procesora oraz pamięci RAM telefonu przy tej samej długości czasu użytkowania wynoszącej 20min.

e) Testy z użyciem baterii, jak aplikacja zachowa się podczas rozładowania urządzenia, co się stanie kiedy nadejdzie połączenie, co ze współpracą aplikacji z wtyczkami np. lokalizacja.

Aplikacja nie wykorzystuje żadnych danych z telefonu użytkownika, tak jak robi to większość z dostępnych gier i aplikacji znajdujących się na Google Play. W trakcie pozostawienia aplikacji uruchomionej, nadal możemy skorzystać bez żadnych problemów z innych aplikacji wykorzystujących



wtyczki, takie jak np. lokalizacja. W wypadku otrzymania połączenia aplikacja działa nadal, do czasu odebrania. Po odebraniu połączenia aplikacja zostaje wstrzymana jednak po zakończeniu rozmowy możemy do niej powrócić i wznowić naszą rozgrywkę.

f) Testy GUI – problemy z poprawnym wyświetlaniem aplikacji na różnych rozdzielczościach

Problemy z wyświetlaniem GUI pojawiają się dopiero w przypadku urządzeń o bardzo małej rozdzielczości ekranu np. 240x320. Oprawa wizualna w rozdzielczości tego pokroju staje się lekko poszarpana a niektóre pojawiające się napisy są mało czytelne (mimo to gra nadal jest grywalna). W wypadku skalowania w górę czyli do rozdzielczości wyższych (nawet 1080x1920, a także 2K), taki problem nie występuje. Zalecaną rozdzielczością jest 1440x2960.

Kod źródłowy: <https://github.com/Karatonik/PAM-RK-River-Raid>

VII. Tabela oceny

Tabela oceny projektu z Programowania urządzeń mobilnych			
lp	Oceniany element	max	punkty
1	Ogólny opis gry (wizja)	1	
2	Analiza dziedziny	2	
3	Specyfikacja wymagań	2	
4	Analiza i projekt:		
	Architektura systemu gry	2	
	Obiektowy model analizy	2	
	Projekt oprogramowania	2	
5	Czytelność kodu źródłowego	2	
6	Podział kodu na moduły/biblioteki/klasy	1	
7	Zgodność gry z oryginałem	6	
8	Dźwięki	4	
9	Muzyka	1	
10	Sterowanie/nawigacja	2	
11	Dodatkowa wersja z rozbudowaną grafiką, dźwiękiem, muzyką	5	
12	Inwencja własna	3	
13	Punkty przyznane od prowadzącego	3	
		40	