Rok. ak. 2021/2022
grupa dziekańska: 6AiSR2
termin realizowania laboratorium: czwartek 8:15

SPRAWOZDANIE Z LABORATORIUM

OPROGRAMOWANIA SYSTEMÓW MIKROPROCESOROWYCH

Autorzy sprawozdania:

1. Elena Gricjuta 228391

2. Arkadiusz Jóźwiak 228401

imię nazwisko nr indeksu

Data wykonania sprawozdania: 27.06.2022

Wersja szablonu 1.2.2

Spis treści

1. Oprogramowanie klasy entertainment na platformę ARosM

1.1. Specyfikacja projektu

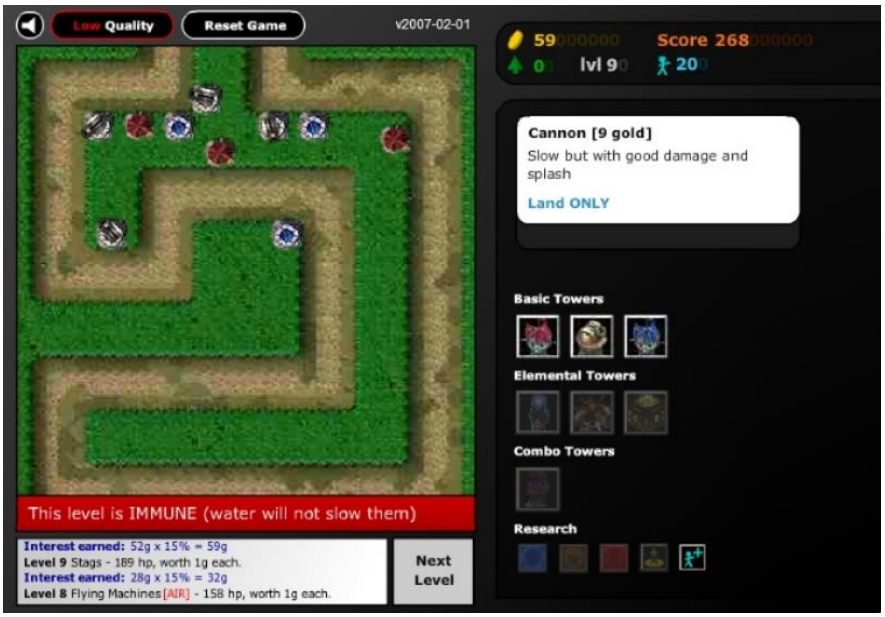
1.1.1 Założenia dotyczące mechaniki rozgrywki

Naszym celem było zaprojektowanie gry pt.: ”Obrona wieży”. Jest to gra strategiczna, w której musimy rozbudowywać swoją bazę przygotowując ją na ataki wrogów. Kluczowy jest tutaj fakt, że te ataki nadchodzą falami i każdy kolejny atak jest silniejszy od poprzedniego. W obronie wieży kluczowe jest również to, że wrogowie mogą się różnić typem a co za tym idzie odpornościami na różne typy broni. Musimy więc naprawdę dobre przemyśleć jak zbudować linię obrony tak aby obrona wieży się powiodła.

1.1.2 Założenia związane z GUI

Jak wyżej opisaliśmy, gra polega na obronie przed wrogami. Mapa powinna się składać z **drogi**, po której idą ci wrogowie, **pola**, na którym gracz stawia swoje wieże, **sklepu** z różnymi typami **wież**, wizualizacji ilości **punktów**, za które można kupić wieże oraz wrogowie.

Na Rysunku 1 został przedstawiony zrzut z przykładowej istniejącej gry.

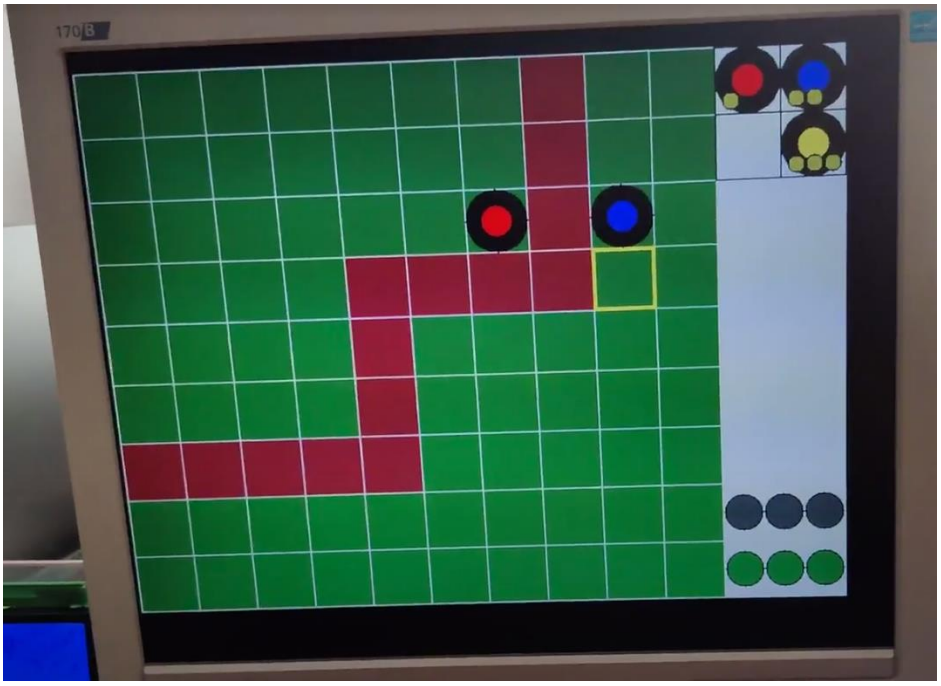


Rysunek 1

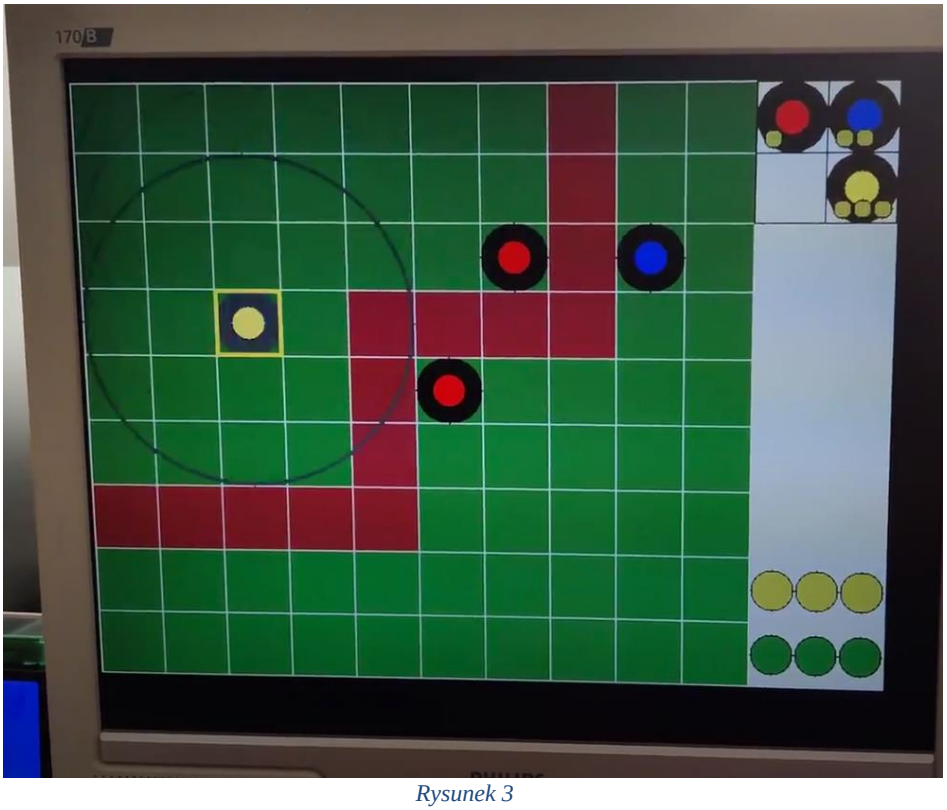
1.2. Dokumentacja oprogramowania klasy entertainment systemu ARosM

1.2.1 Charakterystyka oprogramowania

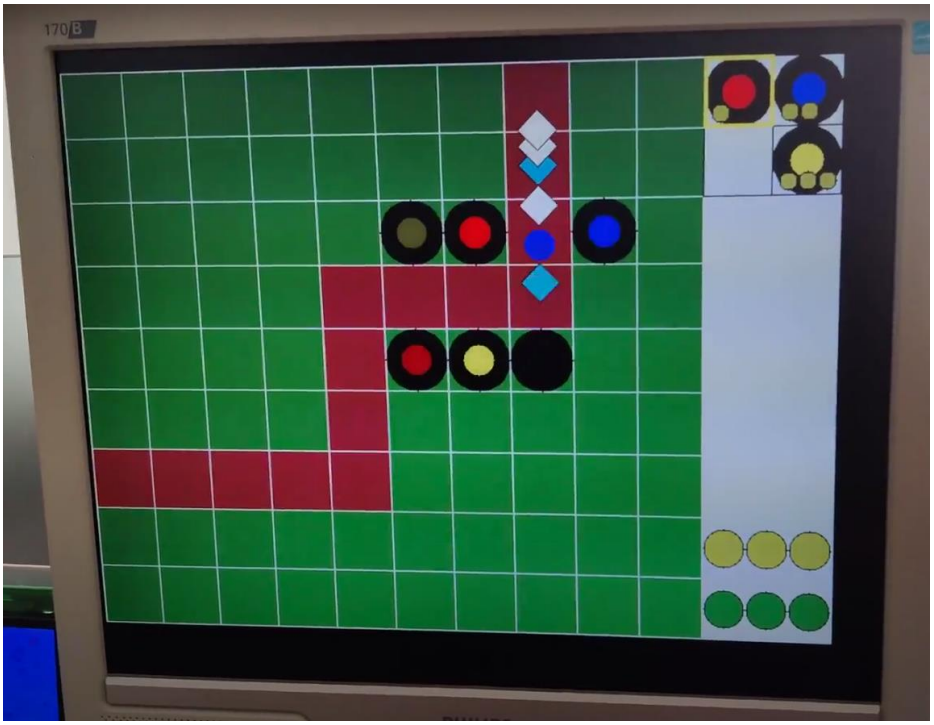
Nasza gra spełniła wszystkie założenia, które zostały opisane powyżej. W naszej grze powstały trzy rodzaje wież: czerwona – podstawowa, niebieska – potrafi dodatkowo spowolnić wroga oraz żółta – szybciej zabija wroga. W dolnym prawym rogu widać ile monetek mamy oraz ile zostało żyć. Każda kolejna fala wrogów jest większa. W tej grze albo przegrywasz, albo uzyskasz taki stan ustalony, że wszystkie wrogowie zostaną zawsze pokonani. Także stawiając wieżę, gracz widzi promień jej zadziałania.
Niżej przedstawiamy kolejne zrzuty z działającej gry.



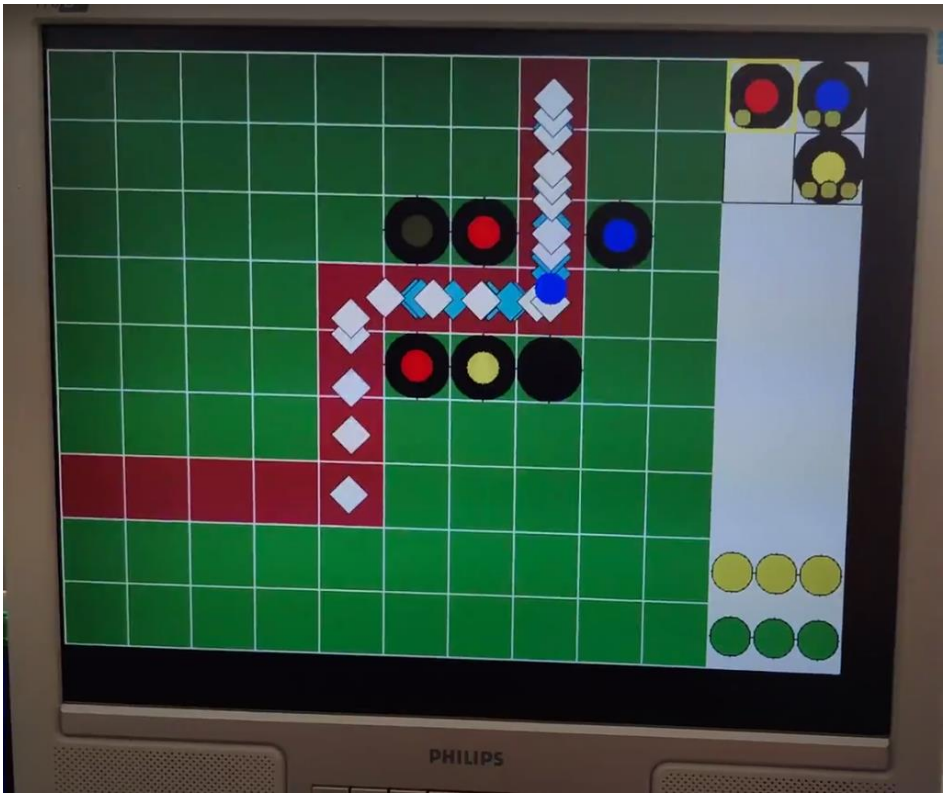
Rysunek 2



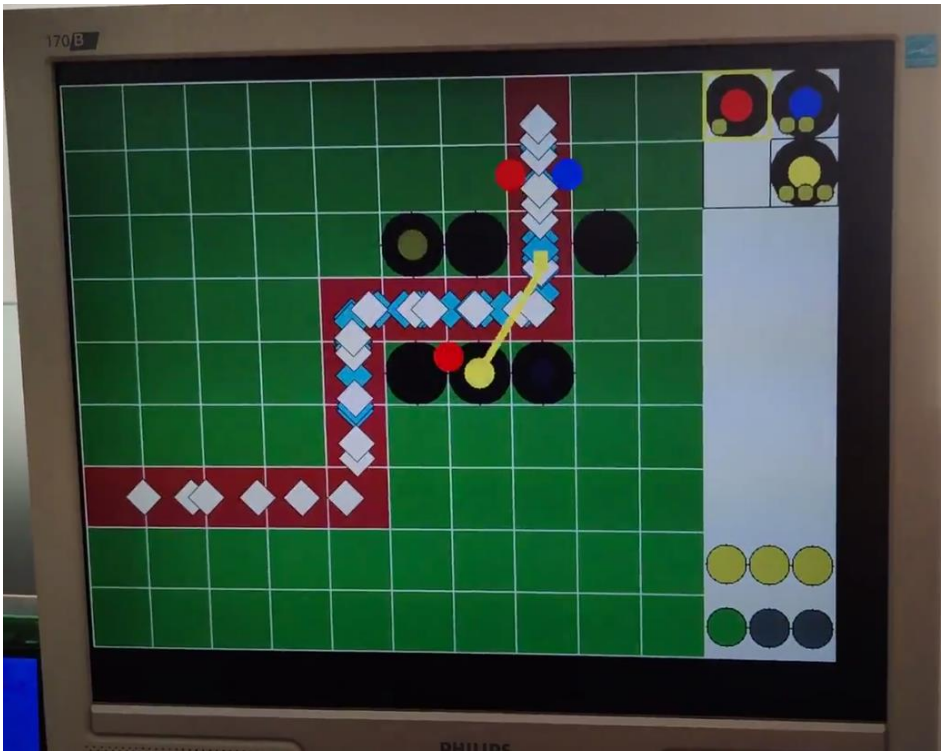
Rysunek 3



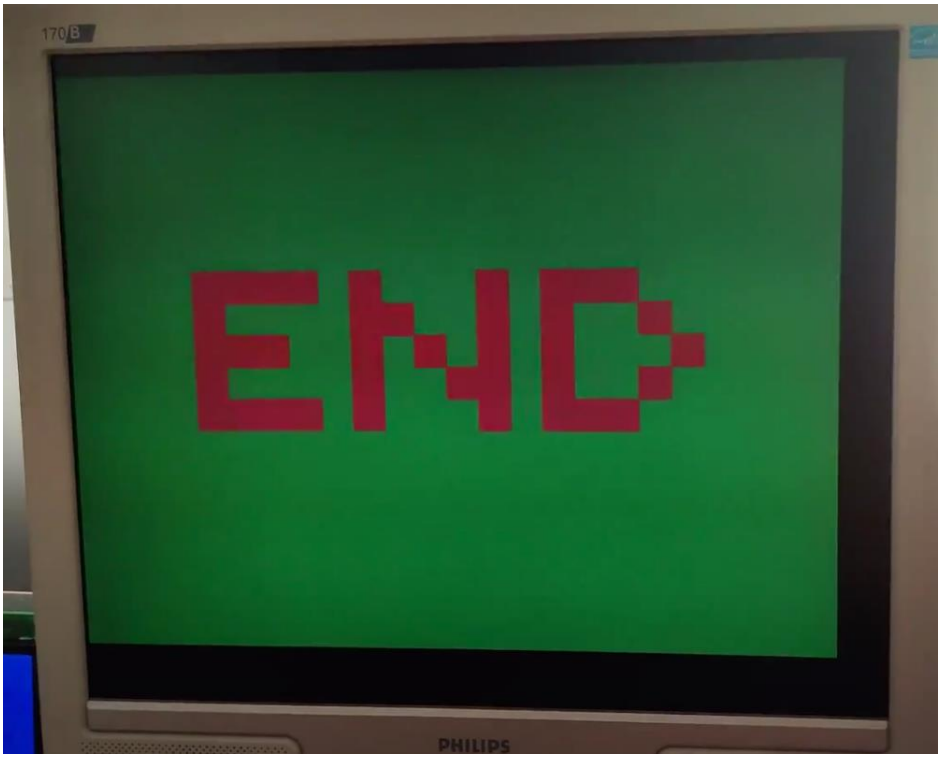
Rysunek 4



Rysunek 5

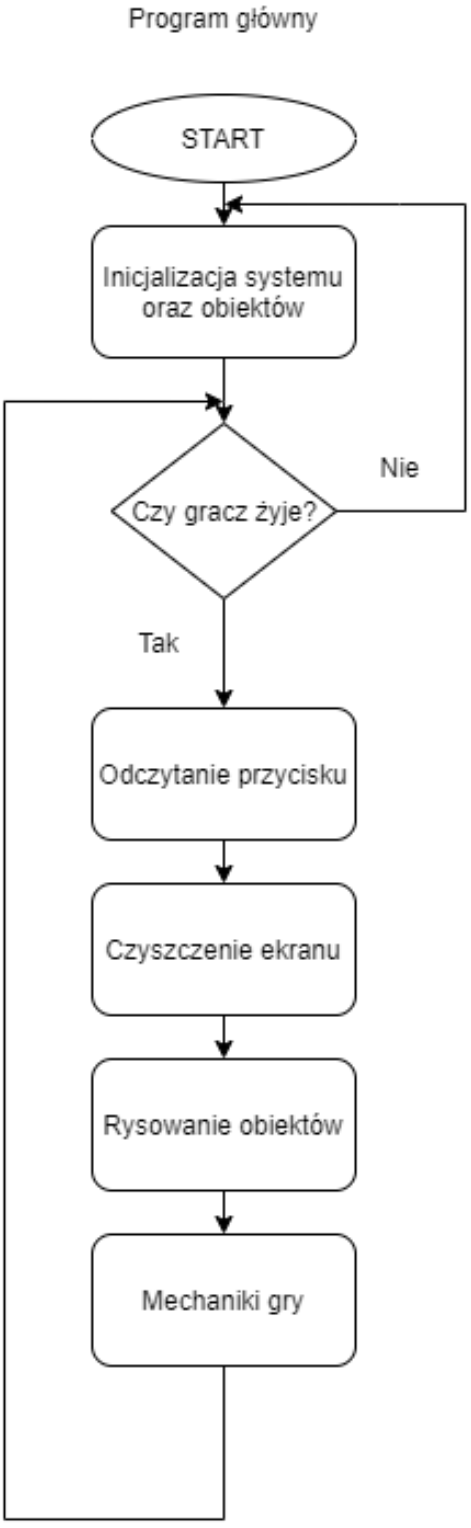


Rysunek 6



Rysunek 7

1.2.2 Schemat blokowe głównych algorytmów



Rysunek 8

1.2.3 Kod źródłowy

```
#define RPiLAB
class Path;
class Enemy;
class Player;
class TurretFire;
class TurretIce;
class TurretBolt;
class Point;

class ProjectileFire
{
public:
    ProjectileFire(int x0, int y0, int
x1, int y1, Enemy *enemy) : x0(x0),
y0(y0), x1(x1), y1(y1), enemy(enemy)
    {
    }
    Enemy *enemy;
    double speed = 30;
    int r = a / 4;
    int color = Color::red;
    int x0, y0, x1, y1;
    double c = sqrt((y1 - y0) * (y1 -
y0) + (x1 - x0) * (x1 - x0));
    int vx = (x1 - x0) * speed / c;
    int vy = (y1 - y0) * speed / c;
    int dmg = 2;

    bool show()
    {
        for (int y = 0; y <= y0 + Y;
y++)
        {
            for (int x = 0; x <= x0 + X
- 2 * a; x++)
            {
                if ((x - x0) * (x - x0)
+ (y - y0) * (y - y0) <= r * r)
                    SetPixel(GRAPH, x,
y, color);
            }
        }
        if (x0 == x1 && y0 == y1)
        {
            enemy->hp -= dmg;
            return false;
        }

        x0 += vx;
        y0 += vy;
        if ((x0 > x1 - speed && x0 < x1
+ speed) && (y0 > y1 - speed && y0 < y1
+ speed))
        {
            x0 = x1;
            y0 = y1;
        }
        return true;
    }
};
class ProjectileBolt
{
public:
    ProjectileBolt(int x0, int y0, int
x1, int y1, std::vector<Enemy *>
enemies3) : x0(x0), y0(y0), x1(x1),
y1(y1), enemies3(enemies3)
    {
    }
    std::vector<Enemy *> enemies3;
    int r = a / 4;
    int color = Color::yellow;
    int x0, y0, x1, y1;
    int dmg = 1;
    bool show()
    {
        Draw::line(x0, y0, x1, y1,
color);
        enemies3[0]->hp -= dmg;
        for (int i = 1; i <
enemies3.size(); i++)
        {
            int xtemp0 = enemies3[i -
1]->xr + a / 2;
            int ytemp0 = enemies3[i -
1]->yr + a / 2;
            int xtemp1 = enemies3[i]-
>xr + a / 2;
            int ytemp1 = enemies3[i]-
>yr + a / 2;
            Draw::line(xtemp0, ytemp0,
xtemp1, ytemp1, color);
            enemies3[i]->hp -= dmg;
        }
        return false;
    }
};
int main(int argc, char *argv[])
{
```

```
    Map map;
    Player player;
    EnemySpawner enemySpawner(&player);
    PathCreator pathCreator;

    SystemInit();
    DataPrepare();
    while (1)
    {
        if (static_cast<Key>(getKey())
!= keyPrev)
        {
            key =
static_cast<Key>(getKey());
            keyPrev = key;
        }
        else
        {
            key = Key::notSet;
        }

        UpdateIO();
        ClearScreen();
        if (player.hp > 0)
        {
        map.forest();
        map.shop();
        pathCreator.show();
        map.gridForest();
        map.gridShop();
        player.show();

        for (auto &e : enemies)
        {
            e.show();
        }
        for (auto &t : turretsFire)
        {
            t.show();
        }

        for (auto &t : turretsIce)
        {
            t.show();
        }

        for (auto &t : turretsBolt)
        {
            t.show();
        }
        for (auto &t : turretsFire)
        {
            t.showProjectiles();
        }

        for (auto &t : turretsIce)
        {
            t.showProjectiles();
        }

        for (auto &t : turretsBolt)
        {
            t.showProjectiles();
        }

        if (START)
        {
            if (!ti_lvl)
            {
                lvl++;
                ti_lvl = tc_lvl;
            }
            enemySpawner.show();
            ti_lvl--;
        }

        if (key == Key::start)
            START = !START;
        player.hpShow();
        player.goldShow();
        }
        else
        {
                map.endScreen();
            if (key == Key::enter)
            {
                player.hp=3;
                player.gold=3;
                START=false;
                lvl=0;
                turretsFire.clear();
                turretsIce.clear();
                turretsBolt.clear();
                enemies.clear();
            }
        }
        usleep(10000);
    }
}
```