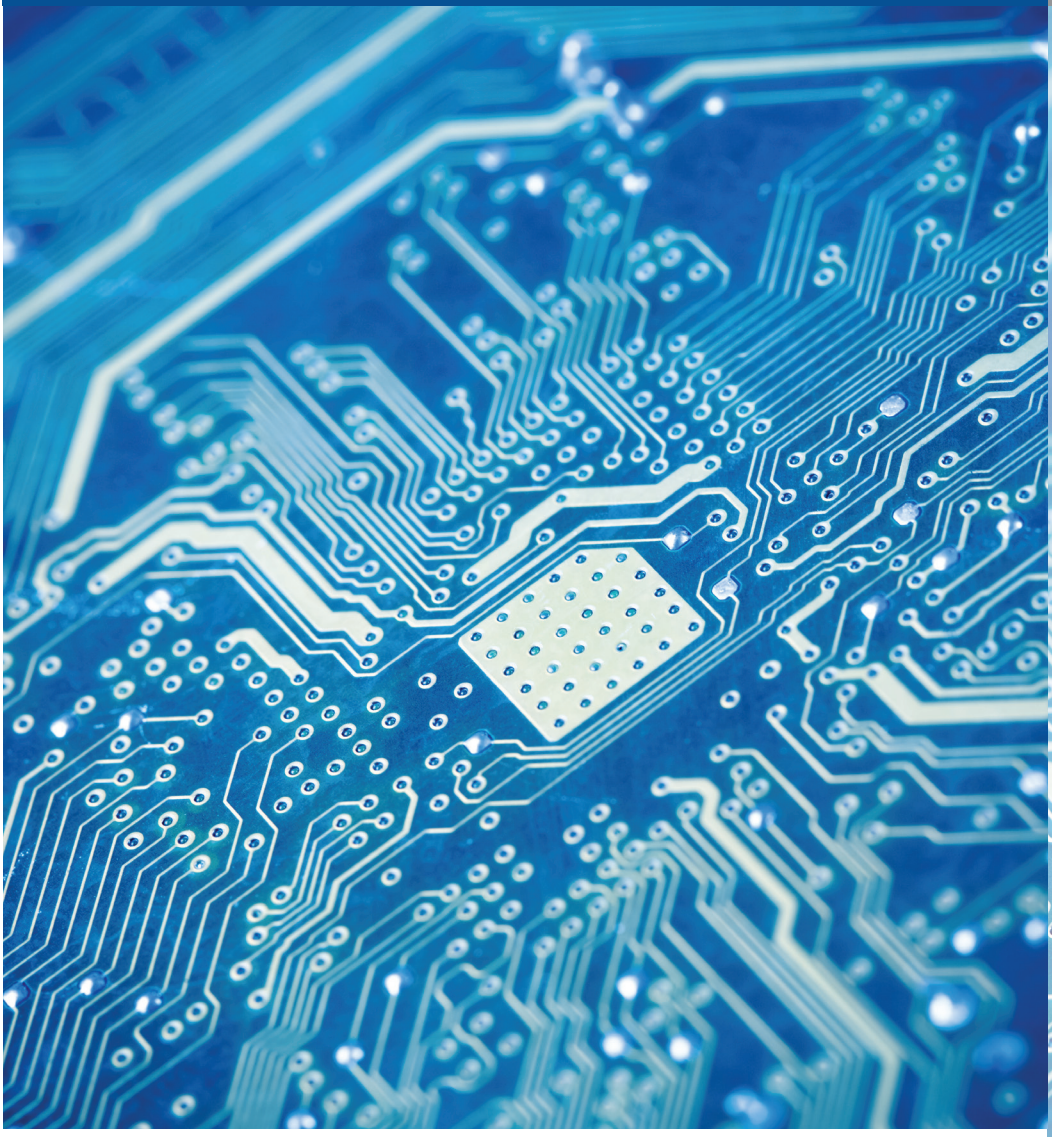
Dolnośląskie Warsztaty Młodych Informatyków

2016

Politechnika Wrocławska



Politechnika Wrocławska

LO nr 3 we Wrocławiu

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

VOLVO

WROCŁAWSKI PARK PRZEMYSŁOWY
DOZAMEL Sp. z o.o. - Zarządca WPP

Patronat honorowy:

Rafał Dutkiewicz, Prezydent Miasta Wrocławia

Prof. dr hab. inż. Tadeusz Więckowski, Rektor Politechniki Wrocławskiej

Dr hab. inż. Zdzisław Szalbierz, Prof. PWr, Dziekan Wydziału Informatyki i Zarządzania PWr

Sponsorzy:

Capgemini

VOLVO

Dozamel

Przewodniczący:

Dr inż. Agnieszka Bienkowska, Prodziekan Wydziału Informatyki i Zarządzania PWr

Mgr Michał Głowacki, Dyrektor III LO we Wrocławiu

Komitet naukowy:

Prof. dr hab. inż. Ngoc Thanh Nguyen (Przewodniczący)

Prof. dr hab. inż. Halina Kwaśnicka

Dr hab. inż. Bogdan Trawiński

Dr inż. Dariusz Król

Dr inż. Marcin Hernes

Dr inż. Marcin Maleszka

Dr inż. Adrianna Kozierkiewicz-Hetmańska

Komitet organizacyjny:

Dr inż. Bernadetta Maleszka

Mgr Dorota Matloch

Dr inż. Rafał Kern

Wstęp

Dolnośląskie Warsztaty Młodych Informatyków (DWMI) są informatyczną konferencją naukową organizowaną przez Wydział Informatyki i Zarządzania Politechniki Wrocławskiej oraz Liceum Ogólnokształcące nr III im. Adama Mickiewicza we Wrocławiu, skierowaną do uczniów szkół ponadgimnazjalnych. Celem **DWMI** jest tworzenie forum naukowego dla ambitnych i zdolnych uczniów zainteresowanych informatyką, którzy zajmują się zagadnieniami badawczymi w tej dziedzinie. Podczas Warsztatów będą oni mogli zaprezentować swoje prace, podyskutować o nich oraz wysłuchać prelekcji na temat aktualnie prowadzonych prac naukowo-badawczych we Wrocławiu i na świecie.

Z przesłanych zgłoszeń Komitet Programowy wybrał 12 referatów do prezentowania podczas Warsztatów. Ich autorzy są uczniami pięciu liceów i gimnazjów z całego terenu Dolnego Śląska. Każdy referujący będzie miał 20 minut aby przedstawić swoje osiągnięcia i odpowiadać na pytania z sali.

Organizatorzy Warsztatów serdecznie dziękują władzom Politechniki Wrocławskiej i Wydziału Informatyki i Zarządzania oraz sponsorom za wsparcie. Serdeczne podziękowanie są skierowane także do młodych naukowców za ich entuzjazm oraz wszystkich uczestników Warsztatów za ich przybycie.

Agnieszka Bienkowska
Michał Głowacki
Ngoc Thanh Nguyen

ALGORYTMY STRATEGII EWOLUCYJNEJ W PROBLEMIE PRZEJŚCIA LABIRYNTU

Zbigniew Drozd
Liceum Ogólnokształcące nr III
im. Adama Mickiewicza we Wrocławiu
ziggi17@wp.pl

Cel:

Projekt ma na celu zbadania zależności szybkości uczenia populacji względem współczynnika mutacji, w selekcji elitarnej w problemie przejścia labiryntu dwuwymiarowego i acyklicznego, w celu skrócenia czasu uczenia populacji bardziej zaawansowanych organizmów w celu rozwiązania bardziej złożonych problemów.

Problem

Przejście labiryntu jest równoznaczne z odwiedzeniem wszystkich jego pól. Do tego celu uczona będzie populacja

Działanie Algorytmu:

Algorytm strategii ewolucyjnej ma pozwolić na znalezienie rozwiązania problemu, w czasie krótszym niż sprawdzenie wszystkich możliwych rozwiązań. Celem jest sprawdzanie, i ulepszanie nieoptymalnych rozwiązań alternatywnych, nazywanych osobnikami.

Implementacja

Labirynt reprezentowany jest przez dwuwymiarową tablicę znaków, gdzie ,'#'' oznacza ścianę a ,' ,' oznacza korytarz po którym osobnik może się poruszać. Genomy osobników zapisane są w tablicy typu całkowitoliczbowego genome[gen][ilość_osobników_populacji].

Osobnik

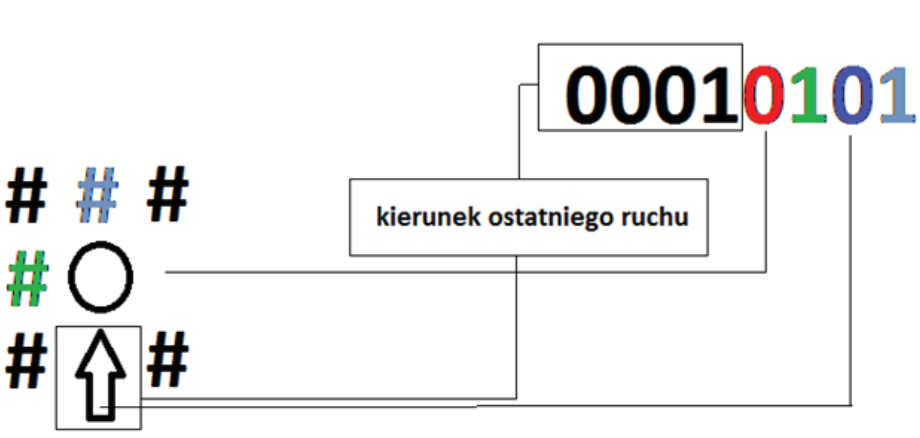
Osobnik określany jest swoim zbiorem genów – genomem, czyli zbiorem zachowań zależnym od stanu jego środowiska. Genom osobników opisywanych w tej procy reprezentowany jest przez tablicę liczb całkowitoliczbowych.

Interakcja Osobnik – Środowisko

Osobniki wyposażone są w dwa zmysły i pamięć, które pozwalają mu na przejście labiryntu. Osobnik wie o stanie ścian, po każdej z jego czterech stron, oraz kierunek z którego ostatnio przyszedł. Te informacje są zapisywane jako maska bitowa, i przesyłane do genomu osobnika

```
short int world(int xpoz, int ypoz, int last)
{
        int input = 0;
        //WT, WD, WL, WR, PT, PD, PL, PR, PU
        if(labirynth[xpoz-1][ypoz] == ,#') input += 1;
        if(labirynth[xpoz+1][ypoz] == ,#') input += 2;
        if(labirynth[xpoz][ypoz-1] == ,#') input += 4;
        if(labirynth[xpoz][ypoz+1] == ,#') input += 8;
        if(last == 1) input += 16;
        if(last == 2) input += 32;
        if(last == 3) input += 64;
        if(last == 4) input += 128;
return input;
}
```

Implementacja w c++ funkcji world zwracającej maskę bitową opisującą stan otoczenia osobnika



Po otrzymaniu danych o środowisku osobnik podejmuje decyzję gdzie pójdzie. Informacja ta jest zapisana jako liczba w genomie w komórce odpowiadającej masce bitowej zwróconej przez funkcję world.

```
int decision = genome[action][num];
```

genome jest tablicą dwuwymiarową, by przechowywać jednocześnie genomy całej populacji.

Decyzja podjęta przez osobnika jest następnie weryfikowana, by sprawdzić czy aktywny gen nie nakazuje wykonania niedozwolonego ruchu (wejścia w ścianę). Jeśli ruch jest dozwolony osobnik go wykonuje, jeśli nie – ginie (dalsza symulacja nie ma sensu, ponieważ sytuacja osobnika po wykonaniu ruchu się nie zmienia - aktywowany jest ten sam gen)

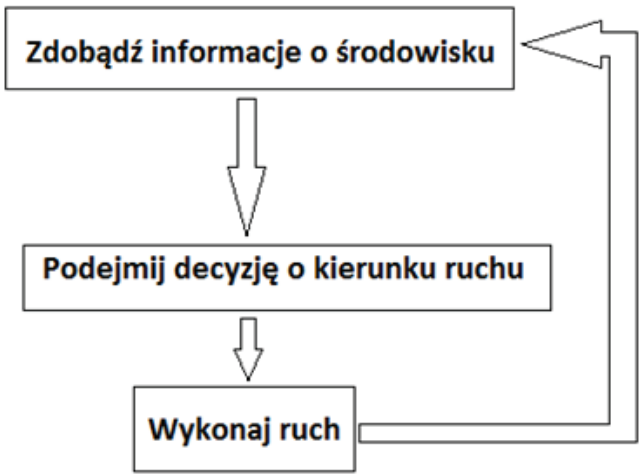
```
bool permission(int xpoz, int ypoz, int action)
{
if(action == 0 && labirynth[xpoz-1][ypoz] != ,#') return true; //idź do góry
if(action == 1 && labirynth[xpoz+1][ypoz] != ,#') return true; //idź w dół
```

```
if(action == 2 && labirynth[xpoz][ypoz-1] != ,#') return true; //idź w lewo
if(action == 3 && labirynth[xpoz][ypoz+1] != ,#') return true; //idź w prawo
return false; // zabij osobnik
}
```

Funkcja logiczna permission przyjmuje 3 argumenty : pozycję osobnika w labiryncie, oraz operację którą chcę wykonać. Jeśli jest ona dozwolona pozwala osobnikowi ją wykonać, jeśli nie – nakazuje go usunąć

```
void execute(int &xpoz, int &ypoz, int decision)
{
        if(decision == 0) xpoz--;
        else if(decision == 1) xpoz++;
        else if(decision == 2) ypoz--;
        else if(decision == 3) ypoz++;
}
```

Funkcja execute, wykonuje decyzję osobnika.



Schemat blokowy działania pojedynczego osobnika

Ocena sprawności organizmu

Gdy organizm wejdzie na wcześniej nieodwiedzone pole, przyznawane mu są punkty przystosowania: 1 punkt za każde pole. Wynik jest podliczany po 2000 ruchów, bądź po wcześniejszym usunięciu organizmu.

Budowanie kolejnej populacji

Gdy wszystkie osobniki z populacji zostaną zasymulowane wybierany jest rodzic do utworzenia kolejnej generacji

Wybór rodzica

Rodzic wybierany jest dzięki selekcji elitarnej – pierwszy w kolejności osobnik no największym współczynniku przystosowania.

Tworzenie kolejnej generacji

Tworzenie N elementowej populacji odbywa się w 3 krokach:

– Przeniesienie genomu wcześniej wybranego rodzica do zerowej komórki w tablicy genomów

– Usunięcia genomów osobników <1, N-1>

– Utworzenia genomów osobników <1, N-1> kopiując genom osobnika o numerze 0 dając każdemu genowi ustaloną szansę na mutację tj. losową jego zmianę.

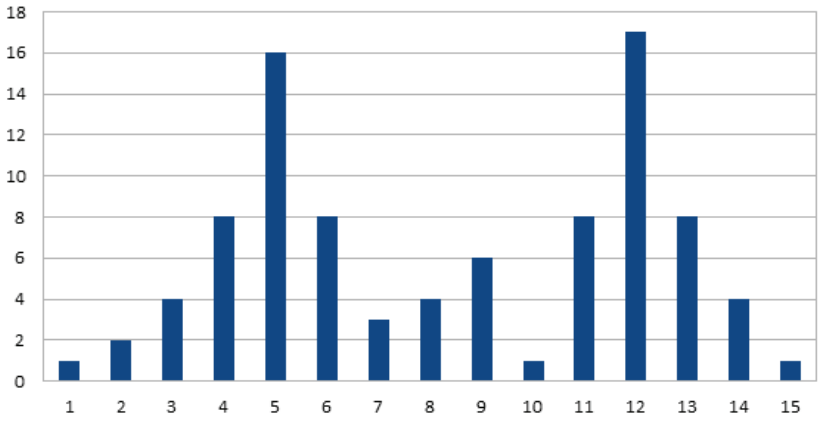
Populacja jest poddawana tej samej próbie co poprzednia, i proces ten jest powtarzany do momentu uzyskania osobnika z maksymalnym współczynnikiem przystosowania

Zalety używania algorytmów strategii ewolucyjnej

Pomijając oczywiste metody rozwiązania problemu przejścia labiryntu (takie jak algorytmy przeszukiwania w głąb czy w szerz) algorytm strategii ewolucyjnej pozwala w bardzo szybkim czasie znaleźć rozwiązanie problemu. Dla implementacji przedstawionej w tej pracy sprawdzenie wszystkich możliwych kombinacji genów (~ 1.34 * 10^154 osobników) zajęło by zdecydowanie za długo. Przy odpowiednio dobranych współczynnikach mutacji, wystarczy symulacja 100000 osobników. Wadą algorytmów strategii ewolucyjnej jest to, że liczba genów osobnika rośnie wykładniczo co do ilości wiadomości o otaczającym środowisku.

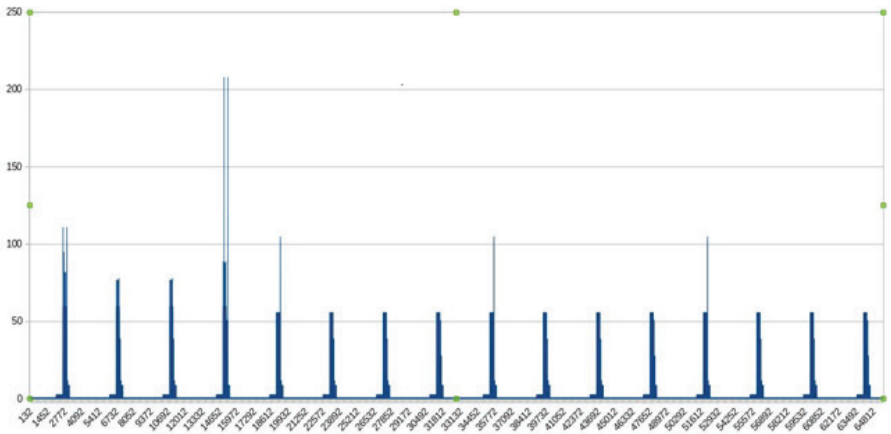
Współczynnik mutacji

Przy tak prostym modelu rozmnażania i selekcji jedynym czynnikiem mającym wpływ na rozwój populacji jest współczynnik mutacji powodujący losowe zmiany w osobnikach. Obrazowo:



Dobór współczynnika mutacji

Przyjmijmy że ten wykres obrazuje wartość przystosowania w zależności od genomu osobnika. Prawidłowo rozwijająca się populacja (na początku rozproszona po całej osi x) po pewnym czasie zacznie się zbliżać do genomu równemu genomowi w polu nr 12. Jednak początkowo wybrane zostanie rozwiązanie nr 5, i współczynnik mutacji będzie zbyt mały to populacja będzie miała duży problem do zbliżenia się do optymalnego rozwiązania. Natomiast jeśli osobniki będą w okolicach rozwiązania numer 12, a współczynnik mutacji będzie za duży, nigdy się do niego nie zbliżą.



Wykres współczynnika przystosowania od genomu dla wszystkich osobników w problemie przejścia labiryntu wg implementacji Arkadiusza Kozdry. Dostarczył te dane na moją prośbę. Z powodu uproszczenia modelu jego osobniki miały 8 (zamiast 256) neuronów, co pozwoliło na utworzenie tego wykresu (możliwych genomów było 65536). Widoczne 2 najlepsze rozwiązania to metody prawej i lewej ręki.

Wnioski

Algorytmy strategii ewolucyjnej pozwalają szybko i skutecznie znaleźć rozwiązania nieskomplikowanych problemów. Biorąc pod uwagę że ilość genów rośnie wykładniczo w stosunku do ilości danych ze środowiska nie należy stosować go do bardziej zaawansowanych problemów. Istnieją jeszcze inne metody selekcji takie jak ruletkowe, czy turniejowe, oraz inne metody hodowania populacji, takie jak na przykład wyspowa, lecz selekcja elitarna, oraz utrzymywanie wszystkich osobników w jednym miejscu jest najprostsze do napisania, i w zupełności wystarcza do rozwiązania problemu przejścia labiryntu acyklicznego.

SZACHY++

Michał Pompa, Michał Wilczak, Dawid Brewiński

Akademickie Liceum Ogólnokształcące
Politechniki Wrocławskiej
michal.pompa.2014@zsa.pwr.edu.pl,
michal.wilczak.2014@zsa.pwr.edu.pl,
dawid.brewinski.2014@zsa.pwr.edu.pl

Wstęp:

Celem naszego projektu było przeniesienie gry w szachy do formy aplikacji konsolowej. Równocześnie rozwinięto umiejętności programowania obiektowo orientowanego w języku C++. Odwzorowano prawie wszystkie mechaniki szachowe, których szczegóły zostaną omówione później. Projekt powstał jako zadanie podsumowujące naukę programowania na zajęciach informatyki. Jednak postanowiono wyjść poza materiał podstawowy i zająć się programowaniem obiektowym.

Opis mechanik gry (algorytmów):

- Roszada – algorytm sprawdza wszystkie warunki potrzebne do prawidłowego przeprowadzenia roszady, tj. czy król nie jest szachowany, następnie czy nie jest szachowane żadne z pól, przez które będzie przechodził oraz czy jest to pierwszy ruch wieży i króla.
- Promocja piona na inną figurę – otwiera się menu wyboru nowej figury.
- Wykrywanie szacha – algorytm sprawdza, czy żadna z figur przeciwnika nie szachuje króla. Nie pozwa-

la królowi wejść w zaszachowane pole(automatyczne cofanie ruchu). Król musi zbić figurę, która go szachuje, uciec, lub zostać zasłonięty przez inną figurę.

- Poprawność ruchu(wzorzec projektowy Strategy) – algorytm sprawdza, czy ruch, który chcemy wykonać jest zgodny z zasadami gry. Sprawdzane jest, czy pole na które chcemy się ruszyć nie jest zajęte oraz czy nic nie stoi na trasie(wyjątkiem jest skoczek).

Dodatkowo gra daje możliwość

- Cofania – każdy ruch jest przechowywany na stosie, dzięki czemu można cofnąć się o dowolną ilość ruchów za pomocą odpowiedniej komendy.
- Zapisu/wczytania gry – grę w dowolnym momencie można zapisać do pliku, a następnie wczytać.
- Zmiany kolorów – plansza jest wyświetlana za pomocą trzech kolorów, które można dowolnie zmieniać.
- Wyświetlanie wiadomości pomocniczej – komenda 'help' wyświetla wiadomość pomocniczą.
- Gra działa na dwóch platformach: Windows i Linux, dzięki zastosowaniu dyrektyw preprocesora do kontroli procesu kompilacji. Aby zmienić wersję na inny system operacyjny wystarczy zakomentować odpowiednią linię w pliku version.info.

W przyszłości zostaną zrealizowane następujące usprawnienia:

- Bicie w przelocie
- Sprawdzanie mata
- Dodanie możliwości gry z komputerem.

Zasada działania:

Podstawą kodu są dwie klasy: Figura, która jest bazą dla wszystkich figur w grze, oraz Board, w której odbywa się zasadnicza rozgrywka.

1. Figura:

Z tej klasy dziedziczą wszystkie figury. Każda przeciąża wirtualną metodę Figura::checkMove(), która sprawdza poprawność ruchu na podstawie przesunięcia. Każda figura przechowuje również swoją reprezentację graficzną, swój kolor i informację, czy figura już wykonała jakiś ruch(dla piona i roszady). W statycznych polach przechowywane są kody kolorów, dzięki czemu można je swobodnie zmieniać.

```
 9 class Figura {
10     friend class Board;
11
12     protected:
13 #ifdef LINUX_VERSION
14         static std::string blackColorCode;
15         static std::string whiteColorCode;
16         static std::string defaultColorCode;
17 #else
18         static int blackColorCode;
19         static int whiteColorCode;
20         static int defaultColorCode;
21 #endif
22         const Color color;
23         const char olaf;
24         mutable bool firstMove;
25         explicit Figura(char='+', Color=UNDEFINED);
26
27     public:
28         explicit Figura(Color=UNDEFINED);
29         Figura(std::string);
30
31         operator std::string() const;
32
33 #ifdef LINUX_VERSION
34         static void setColorCodes(std::string black, std::string white, std::string defaultColor){
35             Figura::blackColorCode=black;
36             Figura::whiteColorCode=white;
37             Figura::defaultColorCode=defaultColor;
38         }
39 #else
40         static void setColorCodes(int black, int white, int defaultColor){
41             Figura::blackColorCode=black;
42             Figura::whiteColorCode=white;
43             Figura::defaultColorCode=defaultColor;
44         }
45 #endif
```

Klasa Figura

2. Board:

```
195 void Board::move(int x, int y, int newX, int newY){
196     x--;
197     newX--;
198     if(x>7||x<0||y<0||y>7||newX<0||newX>7||newY<0||newY>7)
199         throw new RangeException();
200     if(board[x][y]->getChar()=='+')
201         throw new NullField();
202     if(whiteTurn!=board[x][y]->isWhite())
203         throw new TurnException();
204     if((tolower(board[x][y]->getChar())!='s')&&!checkTrack(x, y, newX, newY))
205         throw new TrackNotEmptyException();
```

Walidacja danych w funkcji Board::move()

Ta klasa jest podstawą całej gry. Opakowuje dwuwymiarową tablicę wskaźników na obiekty Figura, reprezentującą szachownicę. Osią programu jest metoda Board::move(); to w niej odbywa się główna część programu, po przekazaniu do niej parametrów wstępnie zwalidowanych w funkcji main().

Są tam sprawdzane podstawowe warunki gry: czy gracz nie próbuje podać współrzędnych pustego pola, czy kolor figury zgadza się z kolorem tury, czy na drodze figury nie stoi inna (wyjątkiem jest skoczek), a następnie czy ruch, który gracz próbuje wykonać jest zgodny z zasadami gry. W tej funkcji składowej implementowane są wzorce projektowe Chain of responsibility oraz Strategy. Metoda Figura::checkMove() sprawdza tylko „standardowe" posunięcia. Ruchy takie jak bicie piona na skos, czy roszada są sprawdzane gdy funkcja Figura::checkMove() zwróci wartość false. Gdy którakolwiek z metod walidacji zawiedzie rzucany jest odpowiedni wyjątek, przechwytywany w funkcji main().
Przebieg programu wygląda następująco:

- Wprowadzenie danych przez użytkownika;
- Sprawdzenie, czy wprowadzone dane nie są komendą gry(np. load, save, help, itp.);
- Wstępna walidacja danych;
- Przekazanie danych do funkcji Board::move();
- Zaawansowana walidacja danych;
- Sprawdzenie, czy zachodzi bicie i ewentualne usunięcie zbitej figury z szachownicy;
- Sprawdzenie, czy król jest szachowany i ewentualne rzucenie wyjątku;

Przy każdym ruchu dodatkowo tworzony jest zapis zmian. Dzięki zaimplementowaniu systemu zapisów na zasadzie stosu, każdy ruch można cofnąć, aż do początku gry.

```
 8 class Board{
 9     struct Save{
10         friend class Board;
11         private:
12             Save *prev;
13             int oldX, oldY, newX, newY;
14             bool oldfm, newfm;
15             bool castling;
16             const Figura *oldF, *newF;
17             Color capturedColor;
18         public:
19             Save(int,int,int,int, const Figura*, const Figura*, bool, Save*, Color);
20     };
```

struktura Board::Save przechowująca zapis zmian

4. Ankieta:

W celach badawczych, wśród użytkowników naszego programu, została przeprowadzona ankieta. Testerzy (w grupie 8 osób) mieli za zadanie, zapoznać się z programem, a następnie odpowiedzieć na następujące pytania:

1. Jak oceniasz naszą grę, w skali od 1-5?

5 – 75%

4 – 25%

3 – 0%

2 – 0%

1 – 0%

2. Czy zasady gry zostały przedstawione w zrozumiały sposób?

TAK – 100%

NIE – 0%

3. Czy używanie programu było skomplikowane?

TAK – 25%

NIE – 75%

4. Czy oprawa wizualna jest przejrzysta?

TAK – 87,5%

NIE – 12,5%

Z przeprowadzonych badań, wynika, że większość użytkowników jest zadowolona z korzystania z naszego programu. Użytkownie go nie sprawiało trudności, a oprawa wizualna nie przeszkadzała w rozgrywce.

5. Podsumowanie:

Dzięki temu projektowi, poszerzono wiedzę, dotyczącą programowania obiektowego w języku C++, oraz poznano zastosowania wzorców projektowych. Przeprowadzone badania pozwoliły nam zorientować się, nad czym należy jeszcze popracować. W przyszłości nasz program zostanie rozwinięty, m.in. o wspomniane wcześniej bicie w przelocie, oraz algorytm sprawdzający mata, ale nie jest wykluczone dodanie innych, ciekawych opcji urozmaicających rozgrywkę.

DRAGONSTONE

Karol Kobiałka, Marcin Standio
Akademickie Liceum Ogólnokształcące
Politechniki Wrocławskiej
karol.kobialka.2014@zsa.pwr.edu.pl,
marcin.standio.2014@zsa.pwr.edu.pl

Celem projektu Dragonstone było rozwinięcie naszych umiejętności w języku C++. Mimo braku większego doświadczenia stwierdziliśmy, iż zajęcie się tworzeniem gry pozwoli nam nie tylko czerpać z pracy ogromną przyjemność, mającej swe źródło w kontakcie z naszym hobby, ale i rozwinąć umiejętności niezbędne podczas pracy na zajęciach z informatyki.

Gry cRPG (computer role-playing games) to elektroniczny odpowiednik popularnych niegdyś gier fabularnych (RPG) rozgrywanych za pomocą kości oraz papieru i ołówka. Pierwsze komputerowe gry fabularne powstały w latach 70 ubiegłego wieku i inspirowane były głównie istniejącymi ówcześnie grami fabularnymi, np. Dungeons and Dragons. Gracz kontroluje bohatera, który posiada oryginalne cechy, wybrane przez grające na początku gry bądź w trakcie rozgrywki. Postać może być rozwijana, może przemierzać świat stworzony przez twórców i wchodzić z nim w interakcję, co zwykle ma na celu rozwój fabuły. Najlepszym przykładem współczesnej gry cRPG będzie rodzima trylogia Wiedźmina na podstawie książek Sapkowskiego - druga i trzecia część są laureatami wielu nagród świata gier elektronicznych.

Projekt Dragonstone narodził się ze wspólnego zamiłowania do gier typu cRPG. Już za młodu umiłowaliśmy sobie ten konkretny gatunek fabularnych przygód wirtualnych postaci i również ten typ postanowiliśmy wziąć na warsztat przy okazji naszego pierwszego wspólnego projektu. Nie był to jedyny powód, dla którego wybraliśmy właśnie gatunek cRPG - w tych grach to nie grafika, a grywalność oraz fabuła grają najważniejszą rolę, co pozwoliło nam oprzeć projekt

w całości o formę tekstową, którą jednak urozmaicaliśmy w trakcie pracy.

Najważniejszym, co chcieliśmy uzyskać w kreacji Dragonstone, to system prawdziwego RPG - gry, w której odgrywa się rolę. Oznacza to wymóg wdrożenia odpowiednich systemów, a wcześniej - stworzenia ich projektów. Do pierwszej wersji gry zaprojektowaliśmy dwa systemy - Arena Fighting oraz Character Development.. Najważniejsze elementy systemu Arena Fighting to przeprowadzenie walki, polegającej na porównaniu statystyk (związanych z systemem Character Development) oraz dobrania odpowiedniego przeciwnika (w zależności od długości rozgrywki). System Character Development odpowiadał za rozwój postaci - zwiększania jej statystyk za złoto zdobywane podczas walk.

Druga wersja przyniosła zdecydowany rozwój wcześniej wspomnianych systemów. Character Development został rozszerzony o system SkillTree. Wymagało to zaprojektowania wszystkich umiejętności, które gracz może zdobywać wraz z rozwojem swojego bohatera oraz przebudowania statystyk, aby umiejętności były od nich nieco zależne. Wprowadzenie SkillTree pozwoliło w pełni oddać kwintesencję gatunku RPG - kreowania unikalnego herosa o własnym zestawie umiejętności oraz statystyk. Możliwości na chwilę obecną jest 256, gdyż każdy bohater ma w ciągu rozgrywki cztery decyzje do podjęcia, każdy polegający na wyborze jednego z czterech dostępnych w tym momencie czarów.

Naturalną konsekwencją wprowadzenia SkillTree była radykalna przebudowa systemu Arena Fighting - w zamian za proste porównywanie statystyk wdrożona została walka turowa. Dało to grającemu możliwość podejmowania decyzji oraz przygotowywania i wykorzystywania strategii podczas walki, co doskonale oddaje charakter Role Playing Game.

Za swój największy sukces uważamy dwa elementy kodu - po pierwsze, turowy system walki, który wprowadzony został do naszej gry przy okazji drugiej wersji, a po drugie - deklarację obiektową potworów, która pozwoliła nam na dowolne operacje na tych obiektach i odwołania przy okazji walk lub innych wydarzeń.

```
//deklaracja postaci
Hero hero;
hero.hp=50; hero.mana=10; hero.max_mana=hero.mana; hero.def=2; hero.max_hp=hero.hp; hero.gold=10; hero.magic_lvl=1;
//deklaracja potworów
Monster mob[7];
mob[0].hp=1; mob[0].max_hp=30; mob[0].atk=3; mob[0].nazwa="Twilight Drake";
mob[1].hp=2; mob[1].max_hp=50; mob[1].atk=4; mob[1].nazwa="Faerie Dragon";
mob[2].hp=4; mob[2].max_hp=70; mob[2].atk=5; mob[2].nazwa="Twilight Guardian";
mob[3].hp=4; mob[3].max_hp=90; mob[3].atk=7; mob[3].nazwa="Hungry Dragon";
mob[4].hp=4; mob[4].max_hp=110; mob[4].atk=9; mob[4].nazwa="Azure Drake";
mob[5].hp=6; mob[5].max_hp=130; mob[5].atk=10; mob[5].nazwa="Volcanic Drake";
mob[6].hp=10; mob[6].max_hp=200; mob[6].atk=12; mob[6].nazwa="Deathwing";
```

Obiektowa deklaracja głównego bohatera, wszystkich spotykanych przeciwników oraz czarów, które gracz ma do dyspozycji pozwala nie tylko na szybką i prostą zmianę statystyk dla balansu gry, niezbędnego do prawidłowego funkcjonowania systemów, o które gra się opiera, ale również na późniejsze odwołania i modyfikacje w ciągu rozgrywki, bez potrzeby głębszej wiedzy na temat funkcjonalności języka. Wszystkie niezbędne zmienne zadeklarowane zostały dla każdego obiektu z osobna. Nie wykorzystujemy specjalnych funkcji zadeklarowanych wewnątrz obiektów, a odwołujemy się do nich bezpośrednio - zdajemy sobie sprawę, iż nie jest to rozwiązanie przyjęte standardowo za poprawne, jednakże w trakcie rozwoju projektu nie byliśmy o tym świadomi. Jest to jedno z pól, które zdecydowanie można rozwinąć w kolejnych wersjach naszej gry RPG.

```
void tura (Hero &hero, Monster &mob, Spell &spell) {
    float dmg = spell.dmg + hero.magic_lvl*spell.bonus;
    if (spell.id != 4 && spell.id != 5 && spell.id != 6) {
        mob.hp=mob.hp-dmg;
        hero.mana=hero.mana-spell.cost;
        cout << "Atakujesz potwora czarem " << spell.nazwa << ", zadajac " << dmg << " obrazen." << endl;
        cout << "Twoja mana: " << hero.mana << ", zycie potwora: " << mob.hp << endl;
        if (mob.hp>0) {
            hero.hp=hero.hp-mob.atk;
            cout << "Potwor atakuje " << hero.imie << ", zadajac " << mob.atk << " obrazen. " << endl;
        }
    }
    if (spell.id==4) {
        hero.hp=hero.hp-(mob.atk-5);
        cout << "Ice Barrier broni Cie przed atakiem potwora! Potwor atakuje " << hero.imie << ", zadajac " << mob.atk-8 << " obrazen. " << endl;
    }
    else if (spell.id==5) {
        if (hero.hp+10>hero.max_hp) hero.hp=hero.max_hp;
        else hero.hp=hero.hp+10;
        cout << "Leczysz sie.";
        if (mob.hp>0) {
            hero.hp=hero.hp-mob.atk;
            cout << "Potwor atakuje " << hero.imie << ", zadajac " << mob.atk << " obrazen. " << endl;
        }
    }
    else if (spell.id==6) {
        cout << "Odbijasz 75% obrazen potwora, czyli " << 3/4*mob.atk << ", jednakze sam przyjmujesz obrazenia." << endl;
        mob.hp=mob.hp-3/4*mob.atk;
        hero.hp=hero.hp-mob.atk;
    }
    spell.curr_cd=spell.max_cd;
    wait_check=false;
}
```

Trudność sprawił nam turowy system walki. Musieliśmy ograniczyć możliwości wylosowania potwora (wybór przeciwnika opiera się na przypadku), a poza tym sprawdzić, czy nasz bohater nie walczy z finałowym przeciwnikiem - w takim przypadku wygrana miała zakończyć przygodę. Oprócz tego, walka musiała opierać się na pętli z możliwościami wyboru kroków. Wyzwaniem było również podawanie użytkownikowi informacji takich jak ilość punktów życia czy liczbę tur, która pozostała do możliwości kolejnego użycia danego czaru.

Gra Dragonstone opiera się na systemie walki na arenie. Podczas rozgrywki wcielamy się w maga-gladiatora, który walczy z różnymi smoczymi przeciwnikami. Mimo zdefiniowanego stylu gry (walka

za pomocą czarów) bohater może być rozwijany w szereg różnych sposobów: gracz do wyboru ma dwanaście różnych umiejętności, z których może opanować cztery (podczas jednego etapu rozwoju można wybrać jedno z trzech różnych zaklęć) oraz rozwijać trzy statystyki określające bohatera (punkty życia, punkty many oraz poziom magii), które gracz „ulepsza" według uznania (i na ile pozwalają mu fundusze zdobyte w grze). Centrum gry to Miasto - stąd gracz może udać się w różne miejsca, z których każde pełni inną, lecz równie ważną w rozgrywce funkcje. W Tawernie gracz za opłatą może zregenerować utracone podczas walk punkty życia i many, w kaplicy gracz rozwija swoją postać (dokonuje wyboru umiejętności oraz za opłatą podnosi statystyki). Na Arenie odbywają się walki. W mieście istnieje również możliwość sprawdzenia statystyk, co daje możliwość planowania przyszłych działań gracza. Każda jedna akcja odbyta w mieście (regeneracja w Tawernie czy odbycie walki na Arenie) powoduje upłynięcie jednego dnia. Co dziesięć dni do puli przeciwników na Arenie trafia nowy smok, a następne umiejętności są zdobywane kolejno co piętnaście, trzydzieści oraz czterdzieści pięć dni. Wymusza to na graczu nieustanną i niewyrównaną walkę z czasem, gdyż gracz musi zdobyć na arenie wystarczającą ilość złota - przygotowanie postaci do walki z coraz potężniejszymi przeciwnikami oraz regenerowania na bieżąco zasobów życia i many są warunkami zwycięstwa.

Walki na arenie odbywają się w systemie turowym: gracz staje do walki z wylosowanym przeciwnikiem z odpowiedniej puli i walczy za pomocą poznanych czarów, zaś przeciwnik zadaje stałe obrażenia (odpowiednie dla rodzaju przeciwnika). Warto dodać, iż moc czarów bohatera rośnie wraz z rozwojem jego statystyk, bowiem wartość obrażeń, jakie zadają przeciwnikowi oblicza-na jest w następujący sposób: suma stałej wartości obrażeń danego czaru oraz odpowiedniego procenta poziomu magii bohatera (obie wartości są odpowiednio przypisane do danych czarów). Czary bohatera obarczone są odpowiednim kosztem punktów many oraz cooldownem (czasem odnowienia, inaczej czasem, po którym umiejętność będzie gotowa do ponownego użytku). Te dwa parametry pozwalają na balans mocniejszych zaklęć przez zwiększenie kosztu i (lub) nadaniem odpowiednio długiego cooldownu, Naturalnym jest, iż podczas walki gracz zużywa całą swoją manę (jedynie w późniejszej fazie gry, gdy bohater jest już w dużym stopniu rozwinięty i trafia na jednego z początkowych przeciwników, gracz ma szansę ukończyć walkę nie zużywając swojego zasobu many). W tym przypadku istnieje możliwość regeneracji, jednakże w tym momencie gra ujawnia swoją „oldschoolową" naturę - podczas regeneracji many, przeciwnik nie przestaje atakować, co znacznie utrudnia starcia na arenie.

Przebieg gry w sześciu zrzutach ekranu.

Na koniec chcielibyśmy wspomnieć o cechach Dragonstone jako gry typu “oldschool”. Najważniejszą cechą owych gier jest fakt, iż “nie wybaczają”, to znaczy błędy popełnione przez gracza (a w szczególności popełnione przez nieuwagę) niosą za sobą konsekwencje i nie ma możliwości na poprawę. W przypadku naszej gry można wyróżnić dwa najczęstsze błędy, które gracz może popełnić: podczas walki w przypadku, kiedy bohater nie posiada wszystkich czarów, próba użycia czaru, którego nie posiadamy, (najczęściej przez pomylenie klawisza) skutkuje brakiem akcji ze strony naszego bohatera, zaś dla przeciwnika tura odbywa się w sposób normalny - zadaje określoną liczbę obrażeń. Drugim błędem jest zła strategia rozwoju w Kaplicy - podczas jednej wizyty (jeden dzień) można rozwinąć statystyki dowolną ilość razy. Często jednak zdarza się, iż gracz podczas szybkiego wydawania poleceń przez przypadek wychodzi z Kaplicy. Aby dokończyć zaplanowany

rozwój musi powrócić, co powoduje upłynięcie kolejnego dnia, a to skutkuje mniejszą szansą na wygranie z ostatnim, finałowym przeciwnikiem.

Podczas tworzenia projektu Dragonstone rozwinęliśmy nasze umiejętności programowania w języku C++. Nauczyliśmy się wykorzystywać globalne zmienne oraz programowanie obiektowe, a po prezentacji również tego, iż błędnie wykorzystaliśmy obie te funkcje języka. W planach mamy wprowadzenie kolejnej aktualizacji do systemu Character Development, dającej grającemu szersze spektrum możliwości kreacji bohatera. Oprócz tego wdrożenie systemu Questing, dającego możliwość graczowi zdobywania nagród za wykonanie zlecanych zadań. Trwają również dyskusje na temat systemu Equipment. Być może wprowadzi on do gry przedmioty używane przez gracza - od broni, zwiększającej obrażenia, aż do magicznych artefaktów, zwiększających statystyki postaci.

„HOW TO ALWAYS WIN TIC TAC TOE" - ANALIZA STRATEGII I ZACHOWAŃ W GRZE KÓŁKO I KRZYŻYK

Michał Matusiak, Jacek Duszenko
Liceum Ogólnokształcące nr III
im. Adama Mickiewicza we Wrocławiu
m.mat1801@gmail.com, jacek.duszenko@gmail.com

Programowanie komputerowego przeciwnika na różnych poziomach trudności.

Wstęp

Celem badań jest omówienie problematyki związanej ze strategiami używanymi w grach. Kółko i krzyżyk, czyli angielskie „Tic Tac Toe" to gra strategiczna rozgrywana przez dwóch graczy na planszy składającej się z dziewięciu pól. Uczestnicy rozgrywki dążą do zajęcia trzech pól w jednej linii, przy jednoczesnym uniemożliwieniu tego samego przeciwnikowi. Osoba, która rozpoczyna grę ma przewagę ruchu, więc łatwiej jest jej wygrać. Osoba rozpoczynająca jako druga może jednak doprowadzić do remisu. Prostota rozgrywki sprawia, że komputer może szybko przewidzieć jak potoczy się sytuacja i już po kilku ruchach wiedzieć, czy dojdzie do remisu, czy do wygranej. Możliwe jest zaprogramowanie komputera tak, by podczas gry niemożliwe było wygranie z nim. W naszych badaniach rozważamy wszystkie strategie i możliwości ruchów w „Tic Tac Toe" tak, aby wyposażyć komputer w różne poziomy trudności.

Ludzie uczący się programowania często otrzymują zadanie napisania gry w kółko i krzyżyk w wersji dla dwóch osób. Nie sprawia to szczególnych trudności, jeśli zna się podstawy programowania w danym języku. My natomiast, oprócz trybu dla dwóch graczy, w naszym projekcie umieściliśmy tryb dla jednego gracza, w którym człowiek rywalizuje z komputerem na trzech poziomach trudności. Wyposażyliśmy komputer w rodzaj prymitywnej sztucznej inteligencji. Dodaliśmy także prosty system statystyk. Całość napisana została w strukturalnym c++. Nie programowaliśmy obiektowo. Gra uruchamia się w konsoli CMD. Zapraszamy do zaznajomienia się z opisem wszystkich części naszego programu.

Tryb dla 2 graczy

Tryb został zamknięty w funkcji void. Na początku następuje sprawdzenie, który gracz wykonuje ruch i na ekran wypisywana jest stosowna informacja. Potem rysowana jest plansza do gry z naniesionymi literami od „a" do „i"(9 liter) na odpowiednich polach. Gdy litera zostanie podana przez gracza, zmienia się na kółko lub krzyżyk. Nad planszą drukowana jest pomocnicza plansza ze współrzędnymi. Następnie sprawdzane są warunki. Jeśli pionowo, poziomo lub ukośnie staną obok siebie trzy takie same znaki (na przykład trzy kółka) to gra się kończy i otrzymujemy informację o zwycięzcy.

Jeśli wszystkie pola zostały zapełnione bez wyłonienia zwycięzcy, to gracz dostanie informacje o remisie. Wszystkie miejsca, w których użytkownik wprowadza dane, zostały opatrzone systemami sprawdzania poprawności. Przykładowo, jeśli użytkownik omyłkowo wpisze losowy ciąg znaków zamiast współrzędnej, to zostanie poinformowany o błędzie i poproszony o wprowadzenie danych ponownie. Przy każdym komunikacie o zakończeniu gry (remis, wygrana, przegrana) jest blok kodu, odpowiadający za statystyki. Więcej o statystykach w odpowiednim akapicie.

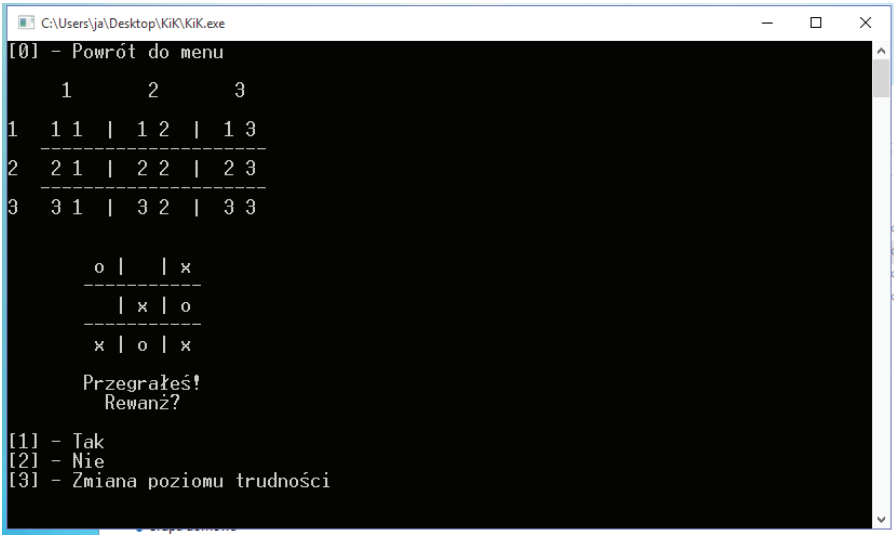
Tryb dla 1 gracza

Ten tryb został zamknięty w kilku funkcjach void i bool. Jego struktura jest dosyć prosta - funkcje odpowiedzialne za poszczególne elementy są odpowiednio uszeregowane w pomocniczej funkcji „głównej" tego trybu, aby nie wprowadzać bałaganu we właściwej funkcji głównej.

Na początku następuje wybór przez gracza poziomu trudności (łatwy, średni, trudny), który to wybór zostanie przekazany do funkcji odpowiadającej za ruch komputera. W tym trybie plansza do gry jest tablicą dwuwymiarową, która po każdym rozpoczęciu gry się zeruje, jak też zeruje się licznik ruchów, który jest pomocny przy sprawdzaniu, czy doszło do remisu. Następnie wyświetlana jest pomocnicza plansza wraz ze współrzędnymi, które trzeba wprowadzić, by móc wykonać ruch (gracz zawsze jest kółkiem). Teraz następuje losowanie, kto ma zaczynać grę z większym prawdopodobieństwem wyniku korzystniejszego dla komputera oraz narysowanie planszy. Pierwszy ruch komputera jest zawsze losowy, a reszta zależy od wybranego przez gracza poziomu trudności, ale o tym później. Następnie ruch wykonuje gracz i jeśli wpisane współrzędne są błędne lub już zajęte, wyświetli się stosowny komunikat. Warto tu nadmienić o zawsze obecnej opcji powrotu do menu (nie tylko w tym trybie), którą możemy w dowolnym momencie przerwać grę. Po ruchu gracza następuje sprawdzenie, czy wygrał, tj. czy obok siebie znajdują się trzy kółka. Następnie ruch wykonuje komputer, a jeśli licznik ruchów jest równy dziewięć, czyli jeśli plansza jest zapełniona, następuje pominięcie kolejki. Kolejnie zostaje sprawdzona możliwość wygranej przez komputer oraz szansa na remis. Jeśli one wystąpią lub jeśli gracz wygra, zostanie on o tym poinformowany (oraz nastąpi uaktualnienie statystyk) i zapytany, czy chce kontynuować rozgrywkę lub zmienić poziom trudności. Od wybranej opcji zależy to, gdzie zostanie przekierowany. W programie obecna jest też funkcja odpowiadająca za poprawność ruchu komputera, tj. czy go w ogóle wykonał.

Na początku rozgrywki gracz ma możliwość wybrania poziomu trudności. Najprostszy, czyli łatwy, zawsze bez względu na sytuację wybiera losowe pole i na nim stawia krzyżyk. Na poziomie średnim i trudnym ta opcja również jest obecna, ale tylko wtedy, gdy żadna z niżej opisanych sytuacji nie ma miejsca. Poziom średni tym się różni od łatwego, że na początku ruchu zawsze sprawdza, czy istnieje możliwość wygrania prze niego lub przez gracza (w tej kolejności) i uzupełnia odpowiedni wiersz bądź kolumnę lub przekątną krzyżykiem. Natomiast poziom trudny jest napisany tak, by nie dało się z nim wygrać. Najpierw następuje sprawdzenie, ile ruchów zostało wykonanych (czyli też kto zaczął) i w zależności od konkretnej sytuacji podejmuje odpowiednie działania. Jak łatwo się domyślić, najbardziej kluczowe jest kilka pierwszych ruchów, tak więc są one bardzo dobrze rozpisane. Jeśli żadna z przewidzianych sytuacji nie zaszła, następuje uzupełnienie do wygranej lub zablokowanie wygranej graczowi (jak na poziomie średnim) bądź wybranie losowego pola. Komputer ma rozeznanie w poziomach trudności dzięki prostym instrukcjom if.

Przykładowy screen z gry



Statystyki

W statystykach znajduje się licznik uruchomień programu, ilość rozgrywek wygranych przez kółko, ilość rozgrywek wygranych przez krzyżyk i ilość remisów. Zasada działania statystyk jest bardzo prosta. Tworzony jest plik tekstowy, który przechowuje w sobie liczby. Jeśli przykładowo doszło do wygranej kółka, to zmienna odpowiadająca za ilość rozgrywek wygranych przez kółko jest inkrementowana i zapisywana w pliku tekstowym. Analogicznie działają pozostałe statystyki. Licznik uruchomień programu działa identycznie, z tym, że fragment kodu, który inkrementuje zmienną odpowiadającą licznikowi został umieszczony na końcu funkcji głównej (int main). Całość została wykonana przy użyciu funkcji z biblioteki fstream.

Main

W funkcji głównej, przy pomocy instrukcji „switch: case" umieściliśmy proste menu. Użytkownik może wybrać spośród czterech opcji: gry dla jednego gracza, dla dwóch graczy, statystyk i wyjścia z programu. Istotną zaletą naszego projektu jest umieszczenie wszystkich trybów gry w osobnych

funkcjach. Teraz wystarczyło tylko wywołać poszczególne funkcje, co znacząco skróciło długość funkcji głównej.

Zakończenie

W naszych badaniach przeanalizowaliśmy na przykładzie prostej gry różne strategie, które mogą doprowadzić do wygranej i zaprogramowaliśmy komputer tak, aby w zależności od wybranego poziomu trudności wybierał strategię. Kółko i krzyżyk jest na tyle nieskomplikowaną grą, że możliwe jest przystępne i niezbyt zaawansowane zaprogramowanie przeciwnika na różnych poziomach trudności.

GRA: STATKI

Eliza Zawisza, Karol Kulawiec
Akademickie Liceum Ogólnokształcące
Politechniki Wrocławskiej
eliza.zawisza.2014@zsa.pwr.edu.pl,
karol.kulawiec.2014@zsa.pwr.edu.pl

CEL:

Stop marnowaniu drzew, czyli zamiana papieru na ekran komputera. Ze względu na popularność gry statki poniższy projekt jest odzwierciedleniem papierowej gry z przysłowiowej szkolnej ławki.

WSTĘP:

W programie istnieją dwa tryby grania, można grać pojedynczo lub w dwie osoby. Grając w trybie jednoosobowym, jednym z graczy jest komputer.
Gracze mają do rozłożenia: 1 statek dwupolowy, 2 statki trzypolowe, 1 statek czteropolowy oraz 1 statek pięciopolowy.
Każdy gracz ma po jednym strzale w kolejce, jednak w wypadku trafienia można wykonać kolejny strzał. Wygrywa ten gracz, który jako pierwszy zatopi wszystkie statki przeciwnika.

OPIS POSZCZEGÓLNYCH ALGORYTMÓW:

- algorytm sprawdzający zatopienie – Na początku określa położenie statku, czy statek jest w pionie, czy w poziomie. następnie idzie pole po polu w jedną stronę, dopóki jest tam trafiony statek,

jeżeli algorytm napotka statek, który jest nie trafiony, to nie zwraca nic, jeżeli trafi na pole wokół statku, to zaczyna działać w przeciwną stronę. Teraz jeżeli znów trafi na pole wokół statku, zwraca zatopienie statku;

- uzupełnianie planszy przez gracza – Gracz podaje współrzędne statku (jego początku), następnie ustala w jakim kierunku statek ma być ustawiony. Operacja ta wykonywana jest dla wszystkich statków w kolejności malejącej (czyli zaczynamy od statku o rozmiarze 5, by na końcu ustalić ustawienie statku o rozmiarze 2);
- uzupełnianie planszy komputera – Algorytm działa podobnie jak w przypadku uzupełniania planszy przez gracza. Losowe współrzędne punktu, w którym zaczyna się statek oraz losowo wybrana orientacja statku (pozioma lub pionowa). Następnie sprawdzana jest możliwość ustawienia statku w ten sposób. Gdyż statki nie mogą na siebie nachodzić i stykać się rogami.
- strzelanie przez gracza- gracz podaje współrzędne punktu, w który chce strzelić, jeżeli trafi, to uruchamia się algorytm sprawdzający, czy statek jest zatopiony, a następnie algorytm strzelania się ponawia, ponieważ gracz otrzymuje dodatkowy ruch. Jeśli gracz nie trafi, algorytm się wyłącza.
- strzelanie przez komputer- Najtrudniejszy i najbardziej skomplikowany algorytm ze wszystkich do tej pory wymienionych. Wisienka na torcie całego projektu. Komputer losuje pole, w które strzela. Jeżeli trafi, to do nowej tablicy wkładane są pola wokół tego punktu, gdzie jest możliwość wystąpienia dalszej części statku i w jedno z nich strzela, jeżeli znów trafi, to w danym kierunku kontynuuje strzelanie, jeżeli w końcu nie trafi, a statek dalej nie będzie zatopiony, to zapamiętuje współrzędne początkowego punktu i w następnej turze strzela w drugą stronę. Jeżeli zatopi statek, to losuje nowe pole, gdzie będzie strzelał, nie uwzględniając pól, gdzie już strzelał oraz pól wokół statków już odkrytych.

PODSUMOWANIE:

Projekt jest syntezą wielu mniejszych programów, które składają się w całość i tworzą idealny związek. W przyszłości planowane jest ulepszenie gry, tak by działała ona na dwóch lub trzech komputerach, aby rozgrywka nie była zbyt monotonna.

GRY LOGICZNE

Daniela Koch, Monika Zemankiewicz, Jakub Filistyński

Akademickie Liceum Ogólnokształcące
Politechniki Wrocławskiej
daniela.koch.2014@zsa.pwr.edu.pl,
monika.zemankiewicz.2014@zsa.pwr.edu.pl,
jakub.filistynski.2014@zsa.pwr.edu.pl

Wstęp

Celem projektu było stworzenie zbioru gierek logicznych pozwalających na sprawdzenie swoich umiejętności oraz porównywanie wyników z innymi użytkownikami. Wkładem własnym jest opracowanie projektu i implementacja algorytmów. W grach sprawdzane są między innymi pamięć, refleks, umiejętność szybkiego myślenia oraz zdolności matematyczne.

Zasady gierek

Każda z gierek ma swoje zasady:

1) Szybkie dodawanie

Na konsoli wyświetlone zostaje kolejno 5 równań. Użytkownik musi w jak najszybszym czasie wpisać poprawną odpowiedź. Za dobre wyniki dostaje punkty, za złe traci.

2) Ciągi

Na konsoli wyświetlone zostają kolejno 3 ciągi. Użytkownik musi znaleźć n-ty wyraz ciągu. Za dobrą odpowiedź otrzymuje punkty.

3) Memory

Gierka ta ma różne poziomy. Na każdym z nich wyświetlane są po kolei cyfry w różnych miejscach ekranu. Użytkownik musi podać kolejność z jaką się pojawiły. Z każdym poziomem liczba wyświetlanych cyfr wzrasta o 1. Użytkownik ma 3 życia. Żeby przejść do następnego poziomu musi wpisać poprawną kolejność cyfr.

4) Marynarzyk

Użytkownik gra z komputerem w marynarzyka, z tym, że komputer z wyprzedzeniem pokazuje swój ruch. Użytkownik ma odpowiednio wygrać lub przegrać. Ważny jest czas reakcji.

5) Drwal

Użytkownik jest 'X'. Jego celem jest omijanie '#'. Porusza się literkami 'a' (lewo) oraz 'l' (prawo). Liczy się liczba o minionych '#' oraz prędkość z jaką będzie się ich unikało.

Opis algorytmów

1) Wybór gry:

Kod podzielony jest na kilka części. Pierwszą z nich jest menu zrobione za pomocą switch'a.

```
29
30 int main(int argc, char** argv) {
31
32   int zadanie;
33   int l=1;
34     srand(time(NULL));
35   while ( zadanie != 0){
36
37     cout << "1- Szybkie dodawanie\n2- Co jest kolejne?\n3- Memory\n4- Marynarzyk\n5- Drwal\n0- Wyjscie\n\n";
38     cout << "Wybierz zadanie: ";
39     cin >> zadanie;
40
41
42     switch (zadanie){
43       case 1:
44         szybkie_dodawanie();
45         break;
46       case 2:
47         ciagi();
48         break;
49       case 3:
50         memory();
51         break;
52       case 4:
53         marynarzyk();
54         break;
55       case 5:
56                 drwal();
57         break;
58     }
59   system("cls");
```

2) Szybkie dodawanie:

W pierwszej grze zostają wylosowane 5 razy 3 liczby, następnie są wyświetlane jako a-b+c. Użytkownik ma za zadanie jak najszybciej podać poprawne odpowiedzi. Przy liczeniu punktów liczy się zarówno liczba poprawnych odpowiedzi, jak i czas.

3) Ciągi:

Stworzone jest 5 funkcji, które generują różne ciągi. Po uruchomieniu gierki zostaje wylosowana kolejność z jaką będą się pojawiały (istnieją trzy możliwości). Następnie w pętli uruchamiane są kolejno ciągi liczb. Za każdym razem jest mierzony czas odpowiedzi.

```
201      start = clock();
202
203    f=rand()%3+1;
204
205      if (f==1){
206         int tab[5]={3, 2, 5, 4, 1};
207      }
208      else if(f==2){
209       int tab[5]={2, 4, 3 , 1 ,5};
210      }
211      else if(f==3){
212       int tab[5]={1, 5, 4, 2 ,3};
213      }
214
215       for (int i=0; i<3; i++){
216         switch (tab[i]){
217           case 1:
218             if(funkcja1() == true){
219                     stop = clock();
220                   cout << "OK ;)\n";
221                   p-=5;
222             }
223               else{
224                   stop = clock();
225                   cout << "Zle :(\n";
226                   p+=6;
227               }
228               t = stop - start;
229             break;
230           case 2:
231             if(funkcja2() == true){
```

Rys 2. Losowanie

4) Memory:

W tej grze wyświetlają się po kolei cyfry w różnych miejscach ekranu, a użytkownik musi podać poprawną kolejność. Aby cyfry nie wyświetlały się w jednej linii tworzone są 2 tablice- jedna, gdzie zapisywane są wylosowane cyfry oraz druga, w której losowane są współrzędne i, o ile nie ma w danej komórce innej wartości, wyświetlane są cyfry z pierwszej tabeli. Wyświetlanie jest oddzielone za pomocą komendy sleep.

```
311  while(i>0){
312    system("cls");
313    cout << "Poziom: " << i-1 << endl;
314    Sleep(1000);
315    int t1[i];
316
317    for(int j=0;j<i;j++)
318      t1[j]=rand()%8+1;
319
320    int t2[8][8];
321
322    Sleep(500);
323    system("cls");
324    Sleep(1000);
325
326    for (int j=0; j<8; j++){
327      for (int k=0; k<8; k++)
328        t2[j][k]=0;
329    }
330
331
332    for (int j=0; j<i; j++){
333          int w = rand()%9;
334          int z = rand()%9;
335
336              if (t2[w][z]==0){
337                  t2[w][z]=t1[j];
338
339                  for(int k=0;k<8;k++){
340                      for (int p=0; p<8; p++){
341                          if(t2[k][p]!=0)
342                              cout << t2[k][p];
343                          else
344                              cout << " ";
345                      }
346                      cout << endl;
347                  }
348                  Sleep(900);
349                  system("cls");
350              }
351              else
352                  j--;
353    }
```

Rys 3. Memory

5) Marynarzyk:

Gierka składa się z trzech funkcji. Każda jest przeznaczona dla jednej z trzech możliwości (nożyce, papier, kamień). W funkcji losowane jest czy gracz musi wygrać bądź przegrać. W funkcji głównej zostaje losowane jedna z trzech funkcji. Całość powtarza się 6 razy.

```
394 void f1(int &p){
395     int a = rand()%2;
396     cout << "Nozyce - 1\nPapier - 2\nKamien - 3\n\n";
397     if(a == 0)
398         cout << "Przegraj\n";
399     else
400         cout << "Wygraj\n";
401     cout<<"\nNozyczki\n";
402     int g;
403     cin >> g;
404     if(a == 0){
405         if(g == 2){
406             cout << "OK ;)\n";
407             p-=5;
408         }
409         else{
410             cout << "Zle :(\n";
411             p+=6;
412         }
413     }
414     else{
415         if(g == 3){
416             cout << "OK ;)\n";
417             p-=5;
418         }
419         else{
420             cout << "Zle :(\n";
421             p+=6;
422         }
```

Rys 4. Marynarzyk

6) Drwal:

Pole, na którym pojawiają się przeszkody jest tablicą o wymiarach 10x2. Losowanie odbywa przeszkód składa się z kilku elementów. Pierwszym jest samo losowanie, czy będzie stawiana na prawy, czy na lewym polu. Aby nie doszło do sytuacji, z której nie dało by się wyjść, jest też druga zmienna. Jeżeli na drodze losowania wyjdzie, że przeszkoda będzie stawiana po przeciwnej stronie niż poprzednio ta zmienna gwarantuje wolne pole.

```
564         system("cls");
565         for(int i = 0; i < 10; i++)
566         {
567             for(int j = 0; j < 2; j++)
568                 cout << t[i][j];
569             cout << endl;
570         }
571         for(int i = 8; i >= 0; i--)
572             t[i+1][0] = t[i][0];
573         for(int i = 8; i >= 0; i--)
574             t[i+1][1] = t[i][1];
575         znak = getch();
576         if(tolower(znak) == 'a')
577         {
578             if(t[9][0] == '#')
579             {
580                 cout << "Przegrales :(\n";
581                 break;
582             }
583             t[9][0] = 'O';
584             p+=20;
585         }
586         else if(tolower(znak) == 'l')
587         {
588             if(t[9][1] == '#')
589             {
590                 cout << "Przegrales :(\n";
591                 break;
592             }
593                 t[9][1] = 'O';
594                 p+=20;
595         }
596         t[0][0] = ' ';
597         t[0][1] = ' ';
598         if(a == c)
599         {
600             t[0][a] = '#';
601             c = a;
602             a = rand()%2;
603         }
```

Rys 5. Drwal

Badania

W celu przetestowania gry udostępniliśmy kod wraz z ankietą. Wyniki prezentują się następująco:
1) 71 % użytkowników oceniło gry na 5, 29 % na 4 (w skali od 1 do 5).
2) Graczom najbardziej podobało się Memory (połowa ankietowanych), jedna trzecia wybrała Drwala, a pozostali Marynarzyka.
3) Jedna osoba stwierdziła, że w Marynarzyku reguły nie były przedstawione w jasny i zrozumiały sposób. Pozostali ankietowani zrozumieli je we wszystkich grach.
4) Prawie wszyscy przebadani chcieliby mieć możliwość porównania swoich wyników z innymi graczami, jednej osobie było to obojętne.
5) Ankietowani najchętniej wróciliby w przyszłości do Memory (83%), Drwala (67%), Marynarzyka (33%). Szybkie dodawanie nie trzymało żadnego głosu.
6) Wszyscy użytkownicy stwierdzili, że punkty przyznawane są w odpowiedni sposób.

.

Z przeprowadzonej ankiety wynika, że użytkownicy najchętniej wybierają gry sprawdzające refleks oraz pamięć. Najmniejszą popularnością cieszą się gry matematyczne. Ważna jest również możliwość porównywania wyników z innymi użytkownikami.

Podsumowanie

W trakcie tworzenia projektu musieliśmy zapoznać się z funkcjami odpowiadającymi za mierzenie czasu. W przyszłości planujemy rozszerzyć nasz program o bazę danych, w której zapisywane będą wyniki poszczególnych graczy.

AUTORSKI JĘZYK SKRYPTOWY PRACUJĄCY Z INTERFEJSEM CGI

Jakub Adamczyk
Liceum Ogólnokształcące nr XIII
im. Aleksandra Fredry we Wrocławiu
ja2015@fejm.pl

Podstawowe cele projektu

- Stworzenie języka programowania realizującego założenia paradygmatu obiektowego i zdarzeniowego.
- Produkcja stron www, rozszerzenie aplikacji sieciowych.
- Udostępnienie intuicyjnych narzędzi do programowania.

Celem projektu jest stworzenie podstaw języka programowania zorientowanego do użycia w praktyce, kiedy matematyczne podstawy informatyki są mniej istotne, przy rozwiązywaniu problemów niewymagających metod matematycznych wprost, programiści na nich nie operują. Powszechne jest stosowanie Frameworków oraz systemów CMS, które znacznie upraszczają pracę, natomiast są mało wydajne. Sytuacja na rynku języków programowania wygląda podobnie – wysokopoziomowe języki operujące na wysokim poziomie abstrakcji.

Jest to wersja wstępna przeznaczona do rozwoju i prezentacji, wobec tego m.in. mechanizmy refleksji, debugowania czy zaawansowanego sterowania przepływem nie są dostępne. Natomiast jest możliwe przygotowanie dodatkowych bibliotek oraz rozszerzenie biblioteki standardowej języka.

Projekt działa z interfejsem CGI. Z racji uruchamiania dla każdego skryptu osobnych instancji proce-

sów, nie jest to działanie wydajne, lecz możliwe jest przepisanie języka jako bibliotek serwera np. Serwera Apache, zwłaszcza że język jest rozszerzalny – funkcjonalność jest zawarta w klasach.

Język wewnętrznie spójny

Jedną z ważniejszych cech języka jest wewnętrzna spójność – jednolitość języka pod względami budowy, interfejsu i nazewnictwa.

Jednolitość i równoważność typów

Abstrakcja projektowanego języka jest nierozerwalnie związana z jej implementacją w C++. Wszystkie dane i struktury mają w nim swoje odbicie – na podstawie deklaracji obiektu budowany jest obiekt w C++, konstrukcje języka (takie jak print, header) są funkcjami, typ danych zmiennej odpowiada typowi danych w C++.

Jednym z wyjątków są typy specjalne (takie jak null, resource, handle) – jest to celowym zabiegiem, który buduje nową warstwę abstrakcji i uodparnia interpretator na wystąpienie błędów niskopoziomowych, które mogłyby się zakończyć np. wyciekiem pamięci czy operacji na nieprawidłowych wartościach.

Wybrane przykłady:

- uchwyty (handle) i referencje oraz wskaźniki: Kompilator nie zezwoliłby na skompilowanie programu z próbą przestawienia referencji na inną zmienną. Przypisanie do wskaźnika null spowodowałoby wystąpienie błędu[CITATION „3”]. Zastąpienie referencji i wskaźników uchwytami uniemożliwia wystąpienie takich błędów.
- null: Definicje literału null w C i C++ się różnią. W C++ literał null jest równoważny z zerem lub zawartością pustego wskaźnika (nullptr_t)[CITATION „4”]. W przypadku możliwości słabej typizacji oraz braku wskaźników takie zastosowanie literału null nie może mieć miejsca.
- resource: W projektowanym języku nie występuje pojęcie strumienia, za pomocą którego interpretator odczytuje pliki, a odczytywanie zawartości baz danych może następować w różny sposób w zależności od sterownika.

Drugim wyjątkiem są specjalne konstrukcje języka (takie jak print, header). Wszystkie są jednak zbudowane w ten sam sposób. Powstały one , by umożliwić operacje niewykonywalne w tym języku (wy-

mienione wyżej konstrukcje umożliwiają wysyłanie danych do standardowego wyjścia interpretatora, które przechwytuje serwer i wysyła do klienta).

Jednolitość nazewnictwa

W niejednolitością nazewnictwa boryka się zarówno tandem C/C++, jak i PHP. Główną przyczyną jest korzystanie z funkcji z C.

Sposoby nazewnictwa w języku:

- Nazwy kilkuwyrazowe zapisywane w kolejności ich występowania w języku angielskim; Brak spacji i innych białych znaków.
- Zmienne i pola: jednowyrazowe nazwy małą literą, wielowyrazowe w formacie CamelCase.
- Stałe: kapitalikami, wielowyrazowe w formacie UNDER_SCORE.
- Klasy i interfejsy: jednowyrazowe wielką literą, wielowyrazowe w formacie StudlyCaps.
- Klasy stanowiące de facto typ danych małą literą i wyłącznie jednowyrazowo (np. int, string).

Jednolite działanie bloków kodu

Niezależnie od tego, czy blok kodu jest zwykłym ograniczeniem instrukcji, czy ciałem struktury kontrolnej, definiuje własny zasięg (więcej w rozdziale bloki kodu).

Podstawowa składnia

Projektowany język jest strukturalny. Opiera się na blokach kodu, które są ograniczone znakami { }. Wszystkie instrukcje niebędące blokiem kodu oddzielane są średnikiem.

Typy proste

Typy proste są obiektami, które udostępniają interfejs pracy podobny do pracy z danymi skalarnymi. Mogą, ale nie muszą być przypisane do zmiennych i stałych. Formalnie każde wystąpienie takiej danej jest typem prostym.

Przykład: *$zmienna=5; // Oba wyrażenia są danymi typów prostych*

Ważne jest, że ciągi znaków powinny być ujęte w cudzysłowy lub apostrofy. Można je stosować naprzemiennie, ale ograniczeniem ciągu znaków muszą być znaki z tej samej pary.

Przykład: *„wyrażenie”;‘wyrażenie’;‘a” „b’;”a’ ,b”;*

Instrukcje, dane, części jednej instrukcji muszą być rozdzielone białymi znakami, lecz rodzaj konkretnego (np. spacja, tabulacja itp.) i ich liczba nie są istotne.

Typy złożone

Typy złożone to zbiory, ciągi i inne struktury. Zbiorem jest prosty obiekt czy prototyp, ciągiem tablica, złożoną strukturą jest m.in. złożony obiekt, klasa. Definicja każdej z nich jest blokiem kodu (czyli musi być ograniczona klamrami). Nie powinno się używać operatora przypisania do ich definicji oraz tak jak po blokach kodu nie wpisuje się po nich średnika.

Przykład
```
$array= {17,24,31,45}// Poprawne, lecz operatorprzypisanianiezalecany.
$object{field:'value',theOther:5}// Poprawne, zalecana składnia.
$array{0:'value',3:'valuf',4:'valug'};// Niepoprawne, średnik na końcu.
```

Komentarze

Rodzaje komentarzy są identyczne, jak w PHP i C (nie ma komentarza w stylu Perla występującego w PHP). Tak jak w PHP i C, zawartość komentarzy nie jest analizowana, więc nie można ich zagnieżdżać.

Przykład
```
// Komentarz jednoliniowy
/* Komentarz zwykły */
```

Bloki kodu

Bloki kodu są wyodrębnioną częścią kodu, które definiują swój zasięg. Działanie jest kaskadowe tj. Bloki kodu zawarte w innych przejmują ich zasięg, natomiast definiowane struktury w danym bloku nie są dostępne dla instrukcji i struktur w blokach go zawierających. Szczególnym przypadkiem bloków kodu są ciała metod, struktur kontrolnych i złożonych obiektów.

Przykład

```
$var1=true;
{
  $var2=true;
  if($var2==true) {
    $var2=false;
    $var3=true;// Istnieją $var1, $var2, $var3.
  }// $var3 nie istnieje, ale $var2 =false.
}// Istnieje tylko $var1
```

Zmienne i stałe

Identyfikator zmiennej musi zaczynać się od znaku dolara:

Przykład:

```
$variable;
```

Stała nie ma prefiksu:

Przykład:

```
constant=5;
```

Dane

Typy proste

Typy proste są obiektami, które służą do operacji na pojedynczych wartościach. W przeciwieństwie do standardowych typów, właściwości obiektów typów prostych obsługuje jedynie język. Wyróżnia się trzy podstawowe typy danych:

- Liczby
- Ciągi znaków
- Wartości logiczne

Wartości logiczne reprezentuje typ numeric, ciągi znaków typ string, wartości logiczne typ boolean.

Język daje duży wybór co do kontroli typów danych (w przeciwieństwie do PHP, który od wersji 7 pozwala narzucać typ, ale nie daje wszystkich możliwości), ale interfejs do tych działań jest bardzo prosty (w przeciwieństwie do C/C++).

- Jeśli zmienna nie będzie deklarowana jawnie, nie przybierze stałego typu, będzie konwertowana automatycznie na dowolny typ w zależności od operacji.
- Jeśli zmienna zostanie zadeklarowana na typ ogólny, będzie automatycznie konwertowana na różne typy szczegółowe w zależności od operacji.
- Jeśli zostanie zadeklarowana na typ szczegółowy, będzie automatycznie konwertowana pod względem rozmiaru.
- Jeśli zostanie zadeklarowana statycznie (wszystkie cechy), konwersja nie będzie możliwa.

Liczby

Definicja:ogólny typ liczbowy	newnumeric$variable

Typy szczegółowe rozdzielają liczby na całkowite i zmiennoprzecinkowe:

Definicja:typ liczb całkowitych	newint[signed][size]$variable
Definicja:typ liczb zmiennoprzecinkowych	newfloat[signed][size]$variable

- signed – znak liczby:
    - unsigned – bez znaku.
    - signed – ze znakiem.
- size – rozmiar zmiennej:
    - 2bit - tylko dla zmiennej całkowitej: wielkość 2 bitów.
    - 4bit - wielkość 4 bitów.
    - 8bit - wielkość 8 bitów.

Operatory

+	dodawanie	dwuargumentowy	$a+$b
-	odejmowanie	dwuargumentowy	$a-$b
*	mnożenie	dwuargumentowy	$a*$b
:lub/	dzielenie	dwuargumentowy	$a:$b $a/$b
%	resztazdzielenia	dwuargumentowy	$a%$b
x++	postinkrementacja	jednoargumentowy	$a++
++x	preinkrementacja	jednoargumentowy	++$a
x--	postdekrementacja	jednoargumentowy	$a--
--x	predekremenracja	jednoargumentowy	--$a
^	potęgowanie	dwuargumentowy	$a^$b

Pola obiektu

value	Wartość zmiennej
sign	Znak
maxValue	Maksymalna wartość zmiennej
minValue	Minimalna wartość zmiennej
type	Szczegółowy typ zmiennej
lenght	Długość zmiennej w bajtach

Ciągi znaków

Definicja:ogólny typznakowy	newstring$variable

Typy szczegółowe rozdzielają ciągi znaków względem strony kodowej:

Definicja:ciągUnicodeUTF-8	newutf-8$variable
Definicja:ciągUnicodeUTF-16	newutf-16$variable
Definicja:ciągUnicodeUTF-32	newutf-32$variable
Definicja:zakodowany ciąg base-64	newbase-64$variable

Domyślnie interpretator operuje na ciągach kodowanych w UTF-8. Jeśli żaden ciąg znaków w projekcie nie jest kodowany inaczej, wszystkie ciągi ogólnego typu będą kodowane w UTF-8.

Operatory

+	konkatenacja	dwuargumentowy	$a+$b

Pola obiektu

alue	Wartość zmiennej
encoding	Strona kodowa
length	Długość w znakach

Wartości logiczne

Definicja	newboolean$variable

Pola

value	wartośćlogiczna
type	wartość:boolean

Operatory

~lub!	zaprzeczenie	jednoargumentowy	~$a lub !$a

Tablice

Tablice pozwalają na przechowywanie wielu danych tego samego typu - może być ogólny lub szczegółowy, a nawet brak – w ostatnim przypadku zostanie ustalony ogólny typ na podstawie pierwszego elementu. Domyślnie elementy są numerowane od zera co jedną wartość.

Przykład:	$array{„First”,”Second”,3} dump$array;

Wyjście:	array$array: {
	Type: string,
	Elements: 3,
	Element[0]: { ‚First’ }
	Element[1]: { ‚Second’ }
	Element[2]: { ‚3’ }
	}

Przykład:	$array[1] =true; $array[5] =true; $array{false,false} dump$array
Wyjście	array$array: { Type:boolean, Elements: 4, Element[1]: {true} Element[5]: {true} Element[6]:{false} Element[7]: {false}

Elementy tablicy można usuwać - używając operatora delete. Podobnie się usuwa całą tablicę.

Proste obiekty

Proste obiekty są strukturami, które mogą przechowywać różne dane (pola) i metody.
Obiekty wspierają słabą kontrolę typów (włącznie z brakiem podania typów).

Przykład:	$object{
	$field = ‚value';
	$other= ‚valuf';
	$theOther= ‚valug';
	theMethod() {
	echo"Themethod!";
	}
	}
	dump$object

Wyjście:	object$object: {
	Elements: 4,
	Fields: 3,
	Methods: 1,
	$field: { ‚value' }
	$other: { ‚valuf' }
	$theOther: { ‚valug' }
	TheMethod: {
	Parameters: {void}
	Returns: {void}
	}
	}

Prototyp

Prototyp jest wzorcem do uzupełniania obiektów. Tak samo jak obiekt może zawierać pola i metody, lecz:

- Może być stosowany przez wiele obiektów.
- Sam nie może być obiektem ani żadną daną (zmienną, stałą).

Do dołączenia prototypu wymagana jest pełna definicja obiektu (nie trzeba podawać typu obiektowego).

Definicja:	prototype %name%{
	...
	}

Przykład:	prototype pattern{
	$name;
	$stamp="Stamp";
	}
	
	$object prototypepattern{$surname;}// Nie zadziała
	dump $object;

Przykład:	prototype pattern{
	$name;
	$stamp="Stamp";
	}
	
	new $object prototypepattern{$surname;}
	dump $object;

Wyjście:	object $object: {
	Elements: 3;
	Fields: 3;
	Methods: 0;
	$name: {null}
	$stamp: { „stamp" }
	$surname: {null}
	}

Złożone obiekty

Złożone obiekty udostępniają wszystkie możliwości wykorzystywania programowania obiektowego języka. Wspierają jedynie silną kontrolę typów (natomiast ich składowe, o ile same nie są złożonymi obiektami, mogą wspierać dowolną). W związku z tym muszą być instancją jakiejś klasy.

Definicja: klasa	class%name%{ ...}

Złożony obiekt może używać również prototypów.

Przykład:	class TypeObject {
	method() {
	returntrue
	}
	}
	
	prototypepattern{
	$array{true,false}
	$object=new TypeObject prototypepattern;
	$object.field=true;
	$object{$foo='bar';$fop='baz'; }
	dump$object;

Wyjście	TypeObject$object: {
	Elements: 6,
	Fields: 4,
	Methods: 1,
	$field: {true}
	$foo: { ,bar' }
	$fop: { ,baz' }
	$array: {
	Element[0]: {true}
	Element[1]: {false}
	}
	}
	Method: {
	Parameters: {void}
	Returns: { 5 }
	}
	}

Rodzaje składowych

W związku z rozszerzaniem złożonych obiektów, istnieje pięć rodzajów składowych, dzięki czemu można zarządzać stanem obiektów oraz ich interfejsem (i oddzielić jedno od drugiego).

Składowa publiczna	public	Widoczna dla wszystkich struktur
Składowa chroniona	protected	Widoczna dla struktur w obiektach klasy pochodnej
Składowa prywatna	private	Widoczna dla struktur wyłącznie w danym obiekcie
Składowa dlapodobiektów	soprotected	Widoczna dla struktur wpodobiektach(kompozycja)
Składowa magiczna	magic	Niewidoczna, specjalny interfejs

Dziedziczenie

Mechanizm dziedziczenia pozwala sprawić, że jeden typ danych może być nadzbiorem lub podzbiorem drugiego. Klasa, której struktury się dziedziczy staje się typem ogólnym, klasa dziedzicząca staje się typem szczegółowym. Język nie umożliwia wielokrotnego dziedziczenia.

Definicja	class %name1%extends%name2%{ ...}

Klasy abstrakcyjne

Podstawową cechą klas abstrakcyjnych jest to, że nie są ostatecznym typem danych. Musi istnieć klasa, która dziedziczy struktury klasy abstrakcyjnej. Klasę abstrakcyjną oznacza się słowem kluczowym abstract.

Definicja:	abstractclass%name%

Klasy finalne

Podstawową cechą klas abstrakcyjnych jest to, że są ostatecznym typem danych. Zabronione jest dziedziczenie struktur klas finalnych. Klasę abstrakcyjną oznacza się słowem kluczowym final.

Definicja:	final class%name%

Klasy statyczne

Podstawową cechą klas abstrakcyjnych jest to, że nie mają swoich obiektów. Zabronione jest tworzenie obiektów klas finalnych. Klasę statyczną oznacza się słowem kluczowym static.

Definicja:	final class%name%

Chcąc uzyskać dostęp do pola lub metody klasy statycznej, należy odwołać się do samej klasy. Klasy statyczne mogą dziedziczyć struktury innych klas statycznych. Mogą istnieć wszystkie rodzaje składowych oprócz magicznych.

Przykład:	static class Info{ privatestring $version = „1.1.80”; echoVersion() { echothis::version } Info::echoVersion();

Interfejs

Interfejs jest zbiorem deklaracji pól i metod publicznych, bez ich zawartości. Oznacza się go słowem kluczowym interface, implementowanie interfejsu w klasie oznacza się słowem kluczowym implements.

Definicja:	interface%name% { ...}

Składowe magiczne

Składowe magiczne stanowią mechanizm, który wykonuje się w określonych momentach (metody) lub pozwalają uniknąć korzystania ze składowych obiektu wprost. Nie można wywołać metod ani pól magicznych.

Metody magiczne

constructor	Konstruktor obiektu.
destructor	Destruktor obiektu.
iterator	Iteratorobiektu.
call	Wykonuje się przy próbie wywołania nieistniejącej lub niewidocznej metody.
get	Wykonuje się przy próbie dostania się do nieistniejącego lub niewidocznegopola.
set	Wykonuje się przy próbie zapisania wartości do nieistniejącego lub niewidocznego pola.
unset	Wykonuje się przypróbie usunięcia niewidocznego lub nieistniejącego pola.
isset	Wykonuje się przy sprawdzeniu na nieistniejącym lub niewidocznym polu czy istnieje.
toString	Wykonuje się przy wywołaniu polecenia traktującego obiekt jak ciąg znaków (np. Operacje na obiekcie jak na ciągu znaków, użycie konstrukcjidump itp.).
sleep	Wykonuje się przyserializacjii uśpieniu obiektu.
invoke	Wykonuje się przydeserializacjiizreaktywowaniuobiektu.

Pola magiczne

Aby użyć pól magicznych należy je umieścić w klasie.

value	To pole zostanie potraktowane jako wartość obiektu.
length	To pole oznacza długość polavalue
type	To pole oznacza typ polavalue

Kompozycja

Kompozycja jest szczególnym przypadkiem agregacji. W przypadku produkcji stron jest szczególnie ważna, ponieważ modelu DOM każdy element jest obiektem, a podelement jest podobiektem. W związku z tym język umożliwia stosowanie mechanizmów kompozycji

soprotected

Słowo kluczowe soprotected oznacza sub-object protected, czyli składową chronioną, ale widoczną w podobiektach. Działa ono tak samo, jak protected dla obiektów danej klasy i klasy pochodnej.

Zarządzanie projektem

Tworzenie spójnych aplikacji w językach skryptowych bywa trudne. Przeszkodami staje się nie tylko nierozróżnianie przez system różnych ról plików, ale też i często nieuprawniona widoczność przy wywołaniu pliku biblioteki bezpośrednio z żądania klienta HTTP. Można się pozbyć tego obosiecznego miecza dzięki wspieraniu wzorców projektowych. Istnieją dwa sposoby na zbudowanie architektury:

- Określenie roli w każdym pliku i dołączanie linków do bibliotek z pliku głównego,
- Dołączanie do pliku link do mapy projektu,
- Zdefiniowanie katalogów projektów i ról skryptów w podkatalogach. Jeśli zostało wysłane żądanie do pliku, który nie jest przeznaczony do wyświetlenia, zostanie automatycznie wysłany kod 403 Forbidden (oraz zostaną podjęte akcje temu towarzyszące). Role mogą dotyczyć:
    - Wyświetlania,
    - Nadrzędności i podrzędności (strona główna, podstrony itp.),
    - Zawartości (dane wykonywalne, informacje do wyświetlenia, definicje klas),
    - Kodów odpowiedzi (np. dany skrypt ma się wykonać po wysłaniu danego kodu odpowiedzi),
    - Danych stron (określone wersje w zależności od żądania klienta itp.),
    - Kilku scenariuszy obsługi (np. w zależności od dostępnych zasobów).

FLAPPY BIRD

Marcin Drutko
Gimnazjum Akademickie Politechniki Wrocławskiej
marcin.drutko.2014@zsa.pwr.edu.pl

Wstęp

Flappy Bird jest to gra na telefony komórkowe, stworzona przez wietnamskiego programistę Dong Nguyena. Zabawa polega na poruszaniu się ptakiem o imieniu „Faby" (w górę i w dół) poprzez dotknięcia ekranu. Zadaniem gracza jest trafienie w otwór pomiędzy dwoma rurami tak, by nie dotknąć żadnej z nich. Celem pracy jest wykonanie własnej kopii tej ciekawej gry.

Charakterystyka gry

Po uruchomieniu gry pojawia się menu główne (rys. 1), zawierające cztery opcje:

rys. 1: Menu główne gry

- Graj
- Opcje
- Autor
- Koniec

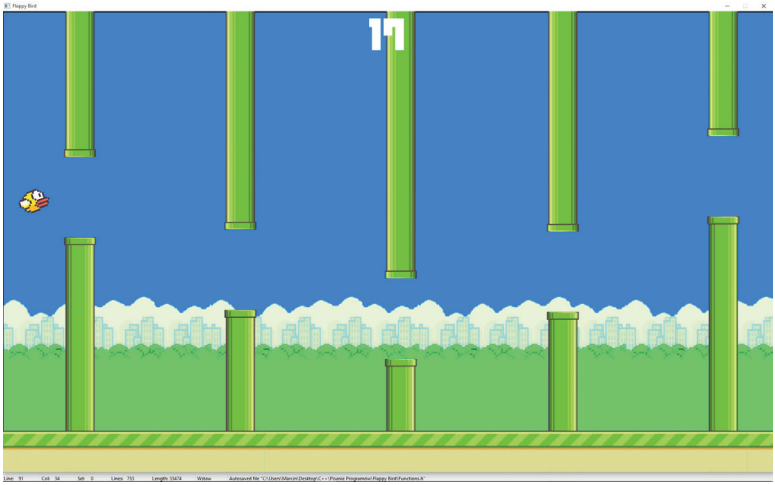
W dalszej części opracowania zostaną krótko scharakteryzowane opcje menu głównego.

1. Graj

Po wybraniu przycisku „GRAJ", gracz przechodzi do rozgrywki (rys. 2).

rys. 2: Początek gry

Akcja gry toczy się podobnie do komórkowego pierwowzoru: zgodnie z życzeniem gracza (przy użyciu klawisza spacji) „Faby" podskakuje (rys. 3).



rys. 3: Rozgrywka

Jeżeli ptak uderzy w ścianę bądź w ziemię, gra się kończy (rys. 4). Pojawia się informacja o tym, ile punktów zebrano, oraz jaki jest najwyższy wynik uzyskany podczas całej rozgrywki. W zależności od zebranych punktów gracz ma możliwość otrzymania medalu.

rys. 4: Koniec gry

Przycisk po lewej stronie pozwala na ponowne uruchomienie gry, zaś po prawej – na powrót do menu głównego.

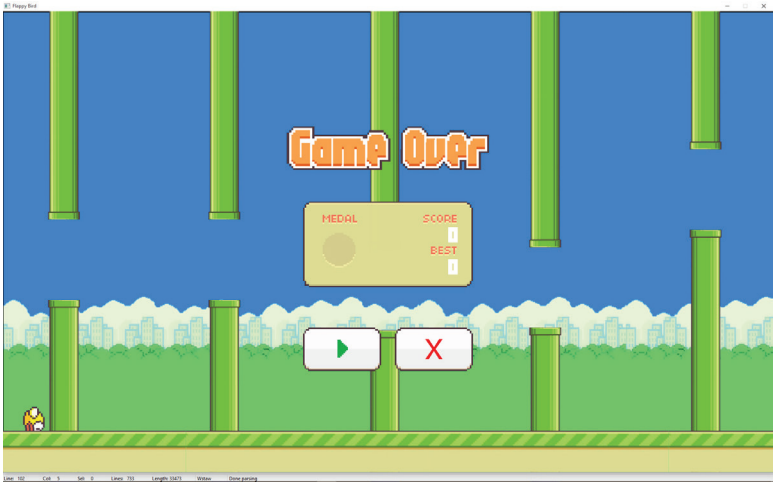
2. Opcje

W opcjach możemy zmienić niektóre właściwości środowiskowe.

rys. 5: Opcje

Pierwszy tekst („DŹWIĘK") informuje o poziomie efektów dźwiękowych. Za pomocą strzałek po prawej stronie możemy zmieniać głośność.

Drugi przycisk pozwala na wyzerowanie aktualnego rekordu gry (rys. 6).



rys. 6: Stan po formatowaniu (wyzerowaniu)

Wciskając przycisk „MENU” na dole ekranu (rys. 5) następuje powrót do menu głównego.

3. Autor

Ekran zawiera informacje nt. autora gry (rys. 7).

rys. 7: Ekran z informacjami o autorze gry

4. Koniec

Za pomocą tego przycisku kończone jest działanie aplikacji.

Dane techniczne

Użyte środowisko programistyczne: C++, Dev-C++ 5.5.1
Wymagania sprzętowe: system Windows XP / 7 / 8 / 8.1 / 10 lub inny dowolny obsługujący pliki o formacie .EXE

Zakończenie

Gra, która wydaje się być łatwą i nieskomplikowaną, jednak może zapewnić wielogodzinną zabawę i rywalizację ze znajomymi. Gra rozwija zdolność szybkiego podejmowania decyzji. Aktualnym celem jest dopracowanie szczegółów gry do jej smartfonowego pierwowzoru.

Wykorzystano materiały ze stron:
http://www.cpp0x.pl/,
http://ww.liballeg.org/a5docs/5.0.10/index.html,
http://delok.free.fr/Boulot1/RM'X%20Vibration/FMOD/fmod361html/HTML/FSOUND.html#Functions,
http://www.stackoverflow.com,
oraz książkę pt. „Język C++ - Szkoła programowania. Wydanie IV" której autorem jest Stephen Prata

EVOLVE - STRATEGICZNA GRA KOMPUTEROWA

Michał Michalak
Liceum Ogólnokształcące nr XII
im. Bolesława Chrobrego we Wrocławiu
michalakmichau@gmail.com

1. Wstęp

Przedstawiona przeze mnie praca jest wstępnym projektem autorskiej gry strategicznej, o tytule „Evolve". Aktualna wersja programu została napisana w języku C#. W celu zachowania pełnej zgodności systemowej, wykorzystałem środowisko programistyczne Unity3D oraz framework mono, umożliwiający integrację programu z platformą .NET na systemie OS X, w którym projekt został napisany.

2. Cel projektu

Celem projektu jest odwzorowanie mechanizmów społecznych oraz ekonomicznych na ekranie cyfrowym zgodnie z decyzjami podjętymi przez użytkownika. Aplikacja ma charakter dydaktyczny oraz rozrywkowy. Służy rozwojowi myślenia taktycznego.

3. Charakterystyka gier strategicznych

Osiągnięcie wygranej w grze strategicznej zależy głównie od taktyki, którą przyjął gracz. Wiąże się to z wykorzystaniem umiejętności planowania i logicznego myślenia. Znaczącą rolę w grach tego typu, odgrywa również wiedza użytkownika z zakresu gospodarki, ekonomii oraz tak jak w mojej grze - potrzeb społeczeństwa. Gry strategiczne można podzielić na gry czasu rzeczywistego oraz te, których akcja oparta jest na systemie turowym. Gry wywodzące się z tego rodzaju, często umożliwiają użytkownikowi bezpośrednią kontrolę jednostek lub ich grup w celu wykonania konkretnej akcji. Fabuła tych produkcji zawiera różne elementy strategiczne takie jak: elementy walki, eksploracji świata, rolnictwa itp. W odróżnieniu od gier ekonomicznych i logicznych, gry strategiczne opierają się na planowaniu w obecności większej ilości graczy i dążą do osiągnięcia wygranej za pomocą wykorzystania taktycznej walki zbrojnej.

4. Przegląd wybranych implementacji komputerowych gier strategicznych

4.1. Seria gier "Civilization"

Seria ta, polega na rozbudowywaniu własnych cywilizacji poprzez rozwój miast, który jest realizowany za pomocą produkcji dóbr i jednostek. Ważną rolę w serii odgrywa również rozwój technologii spo-

łecznych. W ramach "Civilization" powstało 18 produkcji opartych na strategicznym myśleniu, które różnią się dostępnymi scenariuszami, technologiami oraz specyficznymi elementami rozgrywki turowej. Najnowsze wydanie, o tytule: "Sid Meier's Civilization V" odznacza się możliwością handlowania własnymi terenami oraz .

Seria posiada wątki historyczne i kulturowe, które odgrywają ważną rolę w rozgrywce. Każde państwo w grze posiada technologie, budynki oraz umiejętności typowe dla kultury, którą reprezentuje. Dla przykładu, w rozszerzeniu "Nowy wspaniały świat", wydanym 12 lipca 2013 r., na czele Polski stroi Kazimierz Wielki, a strategia wzbogacona jest o szlaki handlowe.

Produkcje "Civilzation" są skomplikowane i wielowarstwowe. Znalezienie optymalnej drogi do zwycięstwa zajmuje bardzo dużo czasu i wkładu pracy gracza. Gracz musi poznać nie tylko zasady walki zbrojnej, lecz także: tajniki dyplomacji, rozwoju technologicznego oraz zależności gospodarcze i ekonomiczne obowiązujące w grze.

Celem gry jest zdobycie władzy nad światem przedstawionym w grze poprzez odpowiednie działania taktyczne.

4.2. Seria gier "Heroes"

Jest to seria gier turowych, z fabułą osadzoną w świecie fantasy. Nie licząc dodatków, od 1995 roku wydano 7 części tej gry. Gracz może posiadać w swojej dyspozycji, aż ośmiu bohaterów, których zadaniem jest zagarnięcie obcych zamków w celu zdobycia władzy w świecie przedstawionym. Różnorodne scenariusze oraz tryby gry gwarantują długą i ciekawą rozgrywkę.

W czasie rozgrywki, użytkownik podczas eksploracji świata zdobywa artefakty, walczy z wrogimi jednostkami oraz potworami, wypełnia zadania poboczne, rozwija swoje zamki i fortyfikacje. Każdy z bohaterów gracza posiada księgę magii, której może użyć podczas walki z wrogami. Do dyspozycji bohatera należy pięć rodzajów stworzeń.

Początkowo, gracz może rekrutować jedynie słabe jednostki, lecz wraz z rozwojem jego imperium oraz ilością posiadanych materiałów do jego dyspozycji dołączą silniejsze opcje.

5. Implementacja gry

"Evolve" jest strategiczną grą czasu rzeczywistego, skierowaną do osób, które interesują się działaniami taktycznymi i ekonomią. Nie jest jednak typowym przykładem gry strategicznej. Zawiera ona

wiele elementów innowacji ekonomicznej oraz sandboxowej. Aplikacja stawia na kreatywność użytkownika oraz różnorodność, szczególnie jeżeli chodzi o wygląd terenu oraz osady. Budynki miejskie pochodzące z pierwszego etapu rozwoju cywilizacji w grze, różnią się znacząco od tych, które zostały wybudowane w późniejszej fazie rozgrywki.

Zadaniem użytkownika jest zadbanie o społeczeństwo, które zostało przydzielone mu na początku rozgrywki. Celem gry jest dojście do najwyższego poziomu rozwoju cywilizacyjnego poprzez umiejętne oraz strategiczne działania. Gracz ma dostęp do szeregu własności określających jego postęp cywilizacyjny(m.in: liczba mieszkańców, środki finansowe, poziom kulturalny społeczeństwa, poziom zadowolenia społeczeństwa, dostępne minerały i zasoby).

6. Specyfikacja funkcjonalna i niefunkcjonalna

6.1. autorski silnik voxelowy

Opracowany przeze mnie silnik voxelowy pozwala na zmianę wyglądu budowli znajdujących się w osadzie w czasie rzeczywistym lub w edytorze. Dzięki zastosowaniu obiektowości języka C# wprowadzenie całkiem nowego elementu architektonicznego, zapewniającego określone profity z poziomu kodu, nie jest trudne. Renderowaniu tysięcy obiektów graficznych związane jest z ustaleniem ich miejsca na mapie i tego, czy są widoczne w zasięgu pola widzenia gracza. Opracowany przeze mnie dynamiczny edytor obiektów pozwala graczowi nie tylko na wybudowanie nowego budynku, lecz także na zmianę jego wyglądu w dowolnym momencie gry. . Pamięć oszczędzana jest poprzez wyświetlanie tylko tych elementów, które nie są przykryte innymi. Teren generowany jest losowo. Dla przykładu - gracz może rozpocząć swoją rozgrywkę w lesie równikowym lub na pustyni.

6.2. Tryb gry multiplayer - wykorzystanie protokołu komunikacyjnego TCP

Osobna aplikacja serwera, która zarządza rozrywką w trybie gry multiplayer decyduje o całej całej rozgrywce. Wszystkie wartości zmiennych oraz obiekty zostają zapisane na serwerze, który posiada własną bazę danych. Połączenie internetową między klientem a serwerem zapewnione jest dzięki protokołowi TCP. W trybie multiplayer to serwer zarządza tym, co ma zostać wyświetlone na ekranie użytkownika. Zajmuje się on kontrolą zdarzeń oraz obsługą wątków i timerów.

Przykład odczytywania komend serwera:

```
IEnumerator ReadFromServer()
{
    reader = new BinaryReader (client.GetStream ());
    writer = new BinaryWriter (client.GetStream ());

    while (true)
    {
        if (client.GetStream ().DataAvailable)
        {
            order = reader.ReadString ();
            string command = order.Split(':')[0];
            MethodInfo methodToDo = this.GetType().GetMethod(command);
            methodToDo.Invoke (this, null);
        }

        yield return null;
    }

}
```

Utworzony obiekt o nazwie "methodToDo" wykona komendę o nazwie, która została wysłana na serwer, np. każe wybudować budynek danego typu w określonym miejscu lub zwiększyć ilość posiadanego złota.

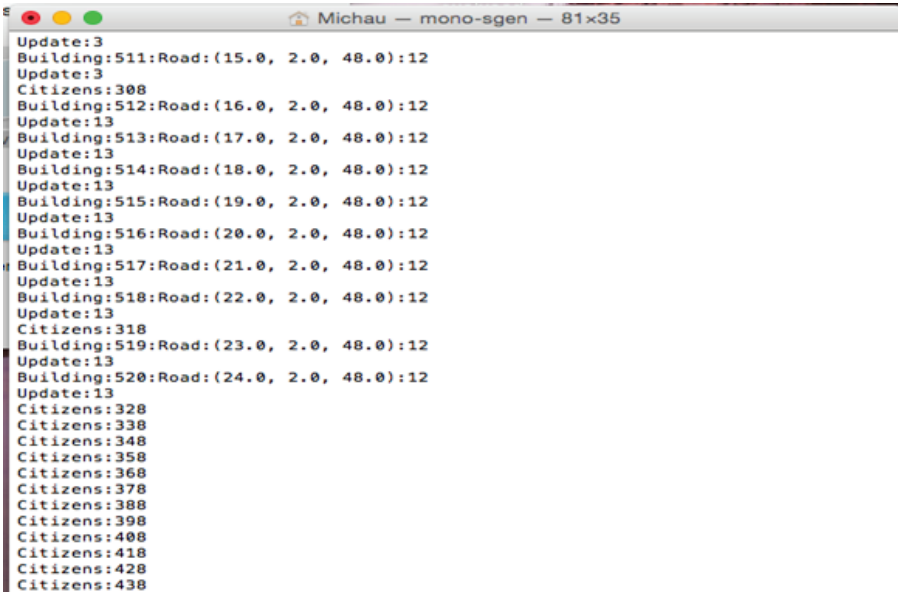
6.3. Mechanizm dynamicznego tworzenia obiektu w trybie rozgrywki internetowej

Wszystko to, co zostało umieszczone na mapie dziedziczy z klasy o nazwie „Building", której zadaniem jest przetwarzanie danych wprowadzonych przez użytkownika w parametry nowo utworzonych, dynamicznych elementów gry. Elementy te, mogą zawierać w sobie zdarzenia powodujące przesłanie danych na serwer. Schemat tworzenia obiektu w trybie gry multiplayer został przedstawiony poniżej:

1. Decyzja o utworzeniu nowego obiektu (np. Ratusza) kierowana ze strony klienta
2. Weryfikacja przesłanego polecenia przez serwer
3. Utworzenia elementu na serwerze
4. Przesłanie danych utworzonego elementu do każdego połączonego użytkownika
5. Odebranie danych przez każdego klienta
6. Utworzenie elementu po stronie aplikacji klienckich

Opisane przeze mnie działania dzieją się w ułamku sekundy, dzięki czemu użytkownik ma wrażenie płynności rozgrywki. Oprócz wspomnianej kontroli zdarzeń serwer zajmuje się również za-

pisem stanu gry do pliku tekstowego. Użytkownik nie musi martwić się o utratę danych, ponieważ wszystko zostaje zapisane po stronie serwera.



Widok serwera:

6.4. Wielowątkowość

Aplikacja nie jest oparta na jednym wątku. Korzysta ona z licznych obiektów klasy Thread, które zajmują się wykonywaniem ogromnej ilości działań matematycznych. Wątki te pracują w tle i dbają o to, aby rozgrywka była płynna na poziomie nie mniejszym niż 30 fpsów. Integracja wątków z budynkami w grze zapobiega zawieszaniu się programu oraz dba o natychmiastową zmianę wartości zmiennych przypisanych do odpowiednich surowców i parametrów.

6.5. Polimorfizm, dziedziczenie

W ramach optymalizacji kodu oraz zwiększenia jego czytelności i wydajności zastosowałem polimorfizm oraz dziedziczenie. Dla przykłady: każda klasa w grze, która reprezentuje obiekt fizyczny, pojawiający się na mapie dziedziczy z klasy o nazwie "Building". Dzięki temu utworzona na serwerze lista obiektów typu Building może przechowywać w swojej kolekcji klasy różnego typu dziedziczącego, w tym: rośliny, animacje, drzewa, budynki. Metody wirtualne, obecne w klasie bazowej, tzw. "rodzica" mogą zostać wywołane przez

każdy obiekt tzw ."dziecka" - klasy, która dziedziczy z bazowej. Zwiększa to znacząco czytelność kodu oraz możliwości odczytu wiadomości zawartych na serwerze. Przykład umieszczony na rycinach poniżej:

```
public class TreeNeutral : Tree
{
    public override int color
    {
        get { return 4; }
    }
    public override string type
    {
        get { return "TreeNeutral";}
    }
    public TreeNeutral(Vector3 locationTree, bool render) : base (locationTree, render){}
    public TreeNeutral(string Id, Vector3 location) : base (Id, location){}
}
public class TreeWhite : Tree
{
    public override int color
    {
        get { return 9; }
    }
    public override string type
    {
        get { return "TreeWhite";}
    }
    public TreeWhite(Vector3 locationTree, bool render) : base (locationTree, render){}
    public TreeWhite(string Id, Vector3 location) : base (Id, location){}
}
public class Tree : Building
{
    public override List <Vector3> blocks { get; set; }
    public override List <Vector3> toRender { get; set; }
    public override int length  { get{ return 4;} }
    public override int width  { get{ return 4;} }
    public virtual int color { get; set; }
    public Tree(string Id, Vector3 location) : base (Id, location){}

    public Tree(Vector3 locationTree, bool render)
    {
        this.render = render;
        location = TerrainClass.SetBlockPosition (locationTree);
        System.Random rand = new System.Random ();
        int h1 = rand.Next (4, 6);
        int h2 = rand.Next (3, 5);
        int createBlock;
        Vector3 globalV3;
        orders = new List<string> ();
        blocks = new List<Vector3> ();
        toRender = new List<Vector3> ();
        //Wood

        for (int n = 0; n < h1; n++) {
            globalV3 = location + new Vector3 (0, n, 0);
            blocks.Add (globalV3);
            orders.Add (":" + Player.ToVector3 ((globalV3).ToString ()) + ":3");
        }

        for (int a = -2; a < 2; a++) {
            for (int b = 0; b < h2; b++) {
                for (int c = -2; c < 2; c++) {
                    createBlock = rand.Next (3, 10);
                    if (createBlock % 3 == 0 || c == 0) {
                        globalV3 = location + new Vector3 (a, h2 + b, c);

                        if (globalV3.z > -1 && globalV3.z < 320 && globalV3.x > -1 && globalV3.x < 320) {
                            blocks.Add (globalV3);
                            orders.Add (":" + Player.ToVector3 ((globalV3).ToString ()) + ":" + color);
                        }
                    }
                }
            }
        }

        CheckBuild ();

    }

}
```

Jak widzimy klasa o nazwie "TreeWhite" dziedziczy zarówno z klasy "Tree" jak i klasy "Building". Dzięki takiemu zastosowaniu może być przechowywana w kolekcji typu "Buidling" oraz przeniesiona na serwer, a ja jako programista w łatwy sposób mogę utworzyć drzewa innego typu poprzez nadpisanie dawnych wartości, rycina niżej:

```
public class TreeBrown : Tree
{
    public override int color
    {
        get { return 7; }
    }
    public override string type
    {
        get { return "TreeBrown";}
    }
    public TreeBrown(Vector3 locationTree, bool render) : base (locationTree, render){}
    public TreeBrown(string Id, Vector3 location) : base (Id, location){}
}
```

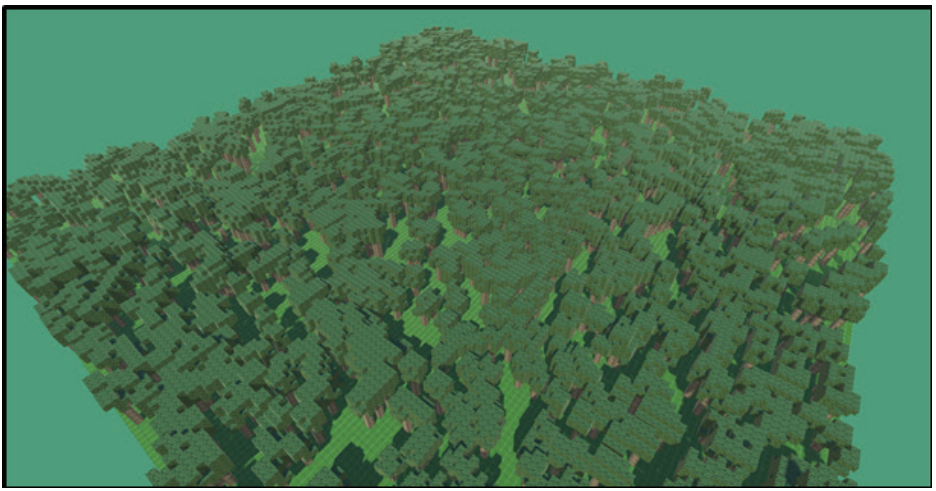
6.6. Szum Perlina

Szum Perlina jest algorytmem generującym pseudolosowe wartości, które imitują naturalne tekstury. Wykorzystano go po raz pierwszy w filmie "Tron" z 1982 r. Stał się przełomowym wydarzeniem w procesie rozwoju proceduralnej grafiki cyfrowej. W grze "Evolve" pozwala na generowanie losowych terenów, opartych na wybranych przez użytkownika wartościach.

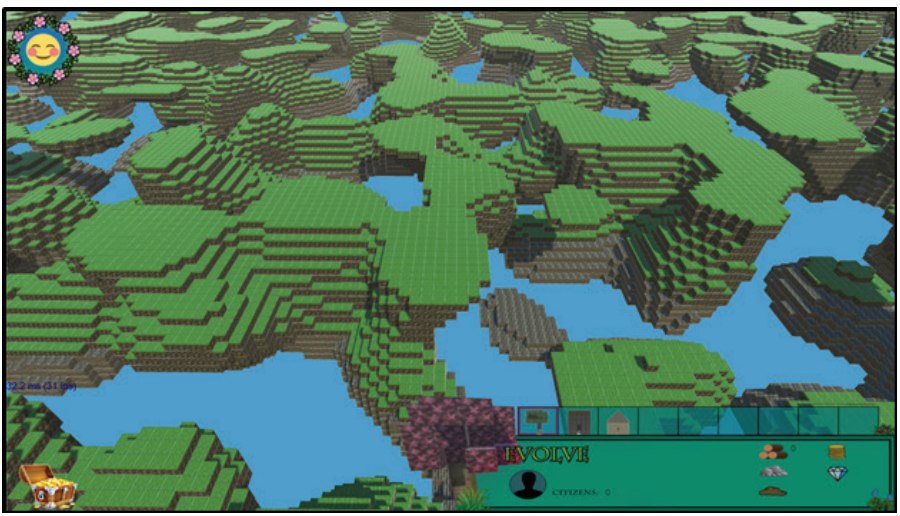
6.7. Wykorzystane tekstury

Tekstury bloków zostały pobrane z darmowych źródeł i przerobione. Zamierzam jednak wprowadzić własne, wykonane przeze mnie w programie graficznym, w podobny sposób jak menu znajdujące się w prawym dolnym rogu:

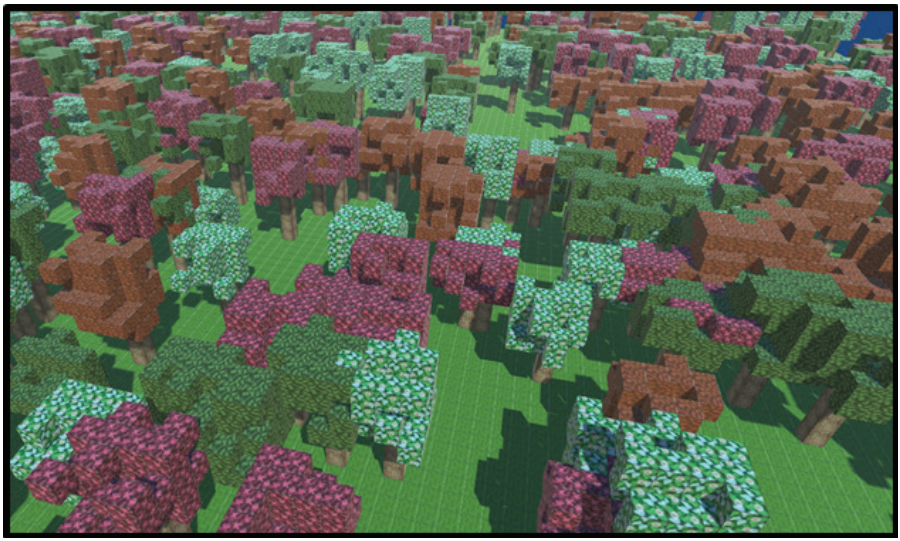
7. Ilustracja przebiegu rozgrywki oraz testy gry



a) Las generowany losowo, o jednym typie drzew na płaskiej powierzchni:



b) Teren pozbawiony flory, na specyficznym, losowym typie powierzchni

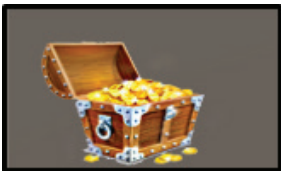


c) Las generowany losowo, o zróżnicowanej florze, na płaskiej powierzchni

Rozgrywka rozpoczyna się w pierwszym etapie rozwoju cywilizacyjnego.
Na ekranie pojawia się interfejs graficzny, na który składa się:

a) poziom zadowolenia społeczeństwa zilustrowany za pomocą obrazka (istnieje wiele różnych stanów)

b) dolny pasek szybkiego wyboru, który zawiera również informacje dotyczące posiadanych surowców, liczby mieszkańców itp.



c) ikona sklepu, w którym użytkownik wykonuje działania związane
z handlem i wymianą surowców

W dowolnym momencie rozgrywki gracz może otworzyć specjalną zakładkę przy pomocy klawisza escape i wrócić do ekranu głównego gry lub ustawień programu.
Poziomy w grze określone są mianem "faz rozwoju cywilizacyjnego", aby przenieść się do następnej fazy rozwoju cywilizacyjnego należy budować odpowiednie konstrukcje zgodnie z własną taktyką, zawierać szlaki handlowe i sojusze lub mierzyć się z innymi graczami.

8. Fazy rozwoju cywilizacyjnego, scenariusze

Fazy rozwoju cywilizacyjnego opierają się zarówno na epokach historycznych jak i wymyślonych przeze mnie okresach. Odznaczają się one swoistymi budynkami, technologiami oraz obiektami, które pojawiają się na mapie. Dojście do ostatniego etapu rozgrywki oznacza zwycięstwo jednak nie kończy ono rozgrywki, gdyż gracz nadal może zawierać ewentualne sojusze lub wrogie traktaty z innymi graczami. Tak naprawdę gra nigdy nie zostanie ukończona, a losy rozgrywki mogą odwrócić się w każdym okresie rozwoju cywilizacyjnego.

1. Antyk
2. Średniowieczne
3. Renesans
4. Późne wieki
5. Współczesność
6. Czasy przyszłe

Gra nie posiada ściśle określonych zasad. To gracz tworzy scenariusz przy pomocy taktyki, którą sam tworzy. Może on stworzyć cywilizację, która zdobywa dobra za pomocą ekspansji i waleczności lub wręcz przeciwnie - cywilizację handlową, o pokojowym nastawieniu.

9. Literatura dotycząca gier strategicznych

10. Podsumowanie i planowane zmiany

"Evolve" jest grą strategiczną o cechach gry ekonomicznej i sandboxowej.
Zapewnia ona użytkownikowi swobodny scenariusz, którego on sam jest kreatorem. Aplikacja kierowana może być za pomocą serwera
oraz programu klienta. Jej zaletami są m.in: obiektowy sposób pisania kodu źródłowego, wielowątkowość, charakter dydaktyczny, luźna fabuła
i możliwość zmiany wyglądu świata przedstawionego, tryb gry multiplayer.
Jej wady to m.in: brak skupienia na detalach - ograniczona ilość budynków dostępnych w grze, serwer włączony zdalnie, który nie został umieszczony jeszcze na maszynie wirtualnej.

Planowane ulepszenia:
- dodanie większej ilości budynków,
- utworzenie nieskończonego świata (w trakcie prac)
- optymalizacja kodu,
- wprowadzenie hodowli zwierząt oraz rolnictwa (w trakcie prac)
- zmiana tekstur

ZDALNY MONITORING DLA KAŻDEGO

Michał Szachniewicz
Akademickie Liceum Ogólnokształcące
Politechniki Wrocławskiej
inco676@gmail.com

Wstęp

Aplikacja pozwala użytkownikowi na zdalny monitoring danego obszaru, wykorzystując jedynie dowolną kamerę oraz mikrokomputer (lub komputer). Oprogramowanie składa się z 2 współpracujących ze sobą modułów. 'Camera' odpowiada za analizę video i wysyłanie informacji do panelu sterowania (modułu 'WebApp'), a ten wyświetla je w czasie rzeczywistym oraz pozwala zmienić podstawowe ustawienia dotyczące kamery.

Całość działa na zasadzie foto-pułapki - jeżeli kamera wykryje ruch, robi zdjęcie z zaznaczonym, wykrytym obiektem, wysyła je na dysk Dropbox, a następnie wyświetla link do zdjęcia na stronie internetowej z centrum dowodzenia, aby użytkownik mógł zidentyfikować zagrożenie i od razu zareagować. Natychmiast po przekazaniu zdjęcia kamera rozpoczyna nagrywanie i analogicznie, po zakończeniu wysyła je na dysk, wyświetlając odpowiedni komunikat dla użytkownika.

Charakterystyka techniczna aplikacji

Demo: http://serene-thicket-83996.herokuapp.com/

Przykłady działania na video: https://youtu.be/KKL9TLTlkLI

Kod: https://github.com/szacho/wmi_proj.git

Wykorzystane dodatkowe moduły:

Python:

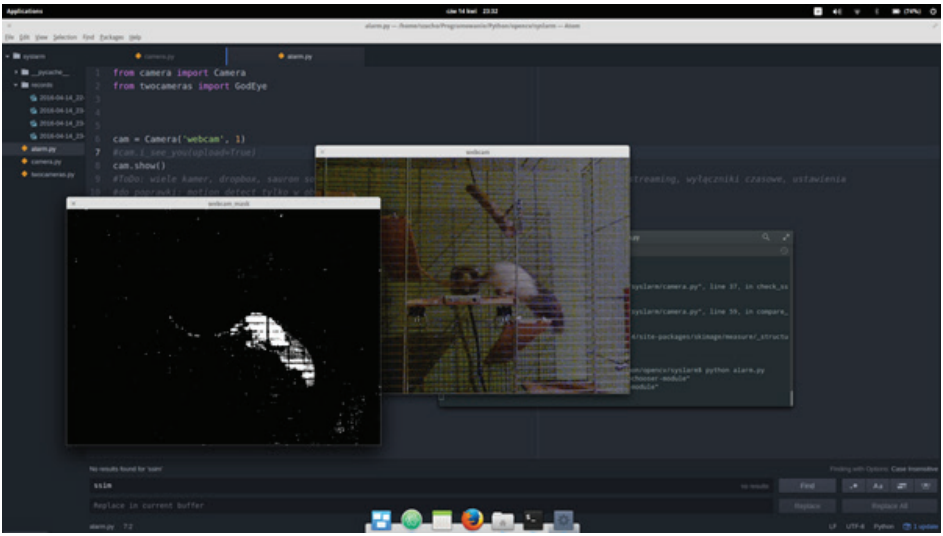
- OpenCV 3.1.0
- scikit-image
- requests, websockets, asyncio
- dropbox
- pyMongo

JavaScript/HTML/CSS:

- MongoDB (mongoose), Express, AngularJS, NodeJS
- websockets
- Bootstrap

Mechanika wykrywania ruchu

Najważniejszym elementem wykrywania ruchu na podstawie samej analizy video jest oddzielenie tła od obiektów ruchomych. Można to zrobić poprzez porównywanie ze sobą klatek następujących w czasie i wyświetlanie różnicy ich pikseli. Jeżeli różnica jest odpowiednio duża, można uznać to za ruch, jednak w ten sposób uzyskujemy słaby efekt po wyświetleniu jej w celu zaznaczenia ruchu na zwykłym zdjęciu, ponieważ zmiany w klatkach widoczne są tylko ułamki sekund. Najlepszym sposobem na rozwiązanie problemu jest skorzystanie z gotowej funkcji OpenCV - createBackgroundSubstractorKNN(). Funkcja ta pobiera argument 'history', który 'zatrzyma' dla nas ruch na odpowiednią ilość czasu.



Rysunek 1 – efekt zastosowania funkcji createBackgroundSubstractorKNN()

Na masce widać ewidentny ruch, ale trzeba to jakoś przekazać do programu. W tym celu użyłem modułu scikit-image i funkcji ssim(), która porównuje dwa czarno-białe obrazy i zwraca ich różnicę w wartości od 0 do 1 (0 jeśli są kompletnie różne - czarny i biały, 1 jeśli są takie same). Potem obraz z maski porównywany jest do samego czarnego tła, dzięki czemu otrzymuję łatwą do przetwarzania i interpretacji wartość. Stąd nietrudno jest napisać funkcję, która wyśle komunikat np. gdy ruchomy obiekt stanowi 10% obrazu lub więcej, ale to nie wszystko - w tej chwili może się zdarzyć, że nasza kamera będzie wykrywać 'stałe' ruchy takie jak np. kołysanie się liści. Aby zignorować ich wpływ na naszą detekcję ruchu należałoby ustawić czułość wykrywania ruchu na sumę czułości jaką mają liście i czułości na jakiej szukamy ruchu. Dlatego program, zanim zacznie szukać ruchu, najpierw się 'nastroi', to znaczy, że przez określony czas będzie liczył średnią różnicę obrazów i uwzględni ją podczas obliczania różnicy obrazów, poniżej której należy zaalarmować ruch. Ponadto funkcja obliczająca czułość bierze pod uwagę wielkość 'stałych ruchów' - im większe są ruchy, jakie chcemy zignorować, tym słabsza czułość jest ustawiania. Pozwala to częściowo zredukować takie problemy jak porywisty wiatr, który raz mocniej, raz słabiej kołysze liśćmi drzew. Cała ta sekwencja wykonuje się nieustannie dla każdej kolejnej klatki obrazu, dzięki czemu reakcja na ruch jest natychmiastowa.

Komunikacja w czasie rzeczywistym

W przypadku wykrycia niepożądanego ruchu na video, praca modułu 'Camera' jest już tak naprawdę zakończona. Po przetworzeniu zdjęcia i wysłaniu plików na dysk resztą zajmuje się aplikacja internetowa, która funkcjonuje w czasie rzeczywistym bez odświeżania strony.

Odpowiada za to nowoczesny protokół WebSocket, który pozwala na przekazywanie między klientem, a serwerem komunikatów w postaci ciągu znaków (w TCP można przekazywać jedynie bajty) oraz na asynchronicznie nawiązywane połączenie, dzięki któremu komunikaty można wysyłać w dowolnej chwili. W klasie Webutils umieściłem wszystkie funkcje odpowiedzialne za komunikację sieciową między kamerą, a panelem sterowania. Funkcja add_event() dodaje zdarzenie do bazy MongoDB oraz konstruuje odpowiednią wiadomość dla webosockets - ustawia temat wiadomości oraz konwertuje obiekt BSON na String.

W wiadomości umieszczona jest także informacja o jej treści, dacie i typie (będzie potrzebny do rozróżniania zdjęć, video i komunikatów). Kiedy serwer otrzyma wiadomość od klienta, rozgłasza ją do wszystkich pozostałych klientów, żeby każdy użytkownik mógł zobaczyć aktualne zdarzenia (plik websockets.js).

HTML/CSS/JS

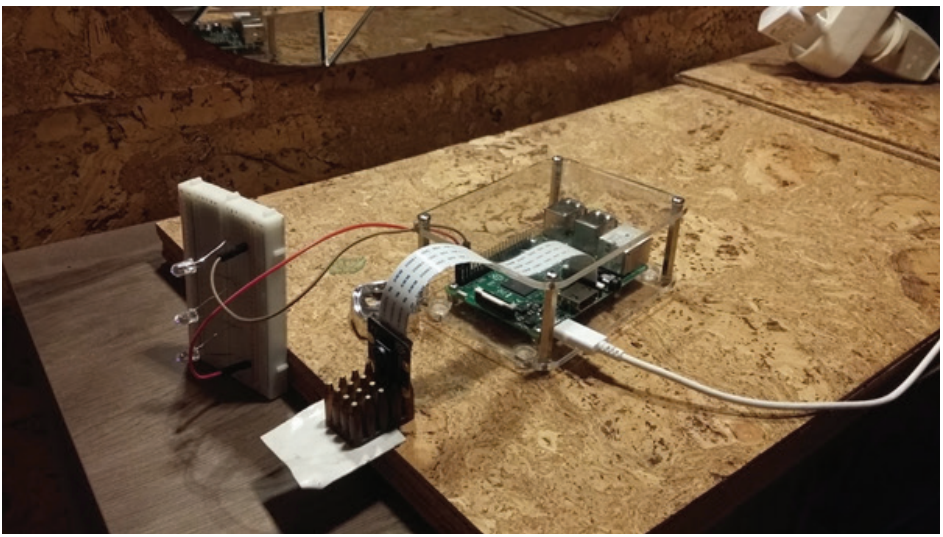
Po rozgłoszeniu komunikatu przez serwer do akcji wkracza HTML6 (AngularJS) - jako klient odbiera i interpretuje wiadomość (ciąg znaków trzeba zamienić na obiekt), a potem uaktualnia szablon strony o nowe informacje, za co odpowiada kontroler 'EventsCtrl' (app.js). Dyrektywa ng-if wyświetla odpowiedni format wiadomości zależnie od tego, czy jest zdjęciem, nagraniem lub komunikatem, ponieważ zarówno link do pliku, jak i treść komunikatu umieszczałem w tej samej komórce. Przycisk czyszczący listę usuwa wydarzenia z bazy danych, ale nie z widoku pozostałych klientów, aby nie irytować użytkowników. Dopiero odświeżenie strony, spowoduje zaktualizowanie się szablonu u innego klienta, więc nikomu podczas przeglądania jej nagle nie znikną wydarzenia.

Drugi kontroler zajmuje się ustawieniami - po naciśnięciu przycisku zapisuje nowe ustawienia w bazie danych z informacją o zmianie ustawień, jaką wychwytuje moduł 'Camera', który co 20 sekund synchronizuje ustawienia i restartuje kamerę, gdy jest potrzeba, aczkolwiek restart może być uruchomiony tylko wtedy, gdy nic ważnego się nie dzieje, tzn. wtedy gdy kamera jest w trybie czuwania, więc

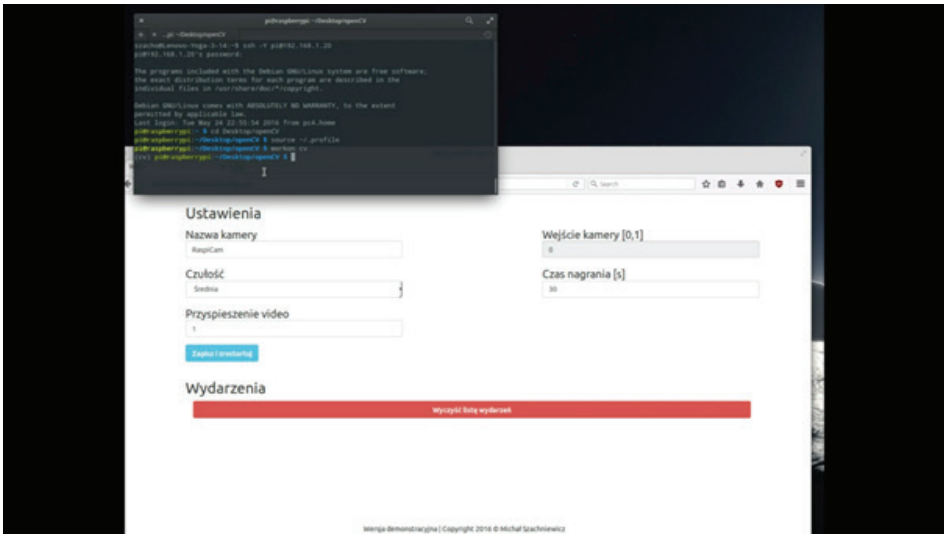
zmiana ustawień nie spowoduje np. błędu przerywającego nagrywanie. Walidacja danych odbywa się także w module 'Camera', aby w razie nieprawidłowości wczytać ustawienia domyślne, zamiast blokować działanie programu. Jeżeli chodzi o same możliwości ustawień to nazwa kamery jest używana w nazwie plików, a czas nagrania podzielony przez przyspieszenie jest równy wyjściowej długości pliku video - warto zmniejszyć objętość plików trzymanych w chmurze.

Eksperyment

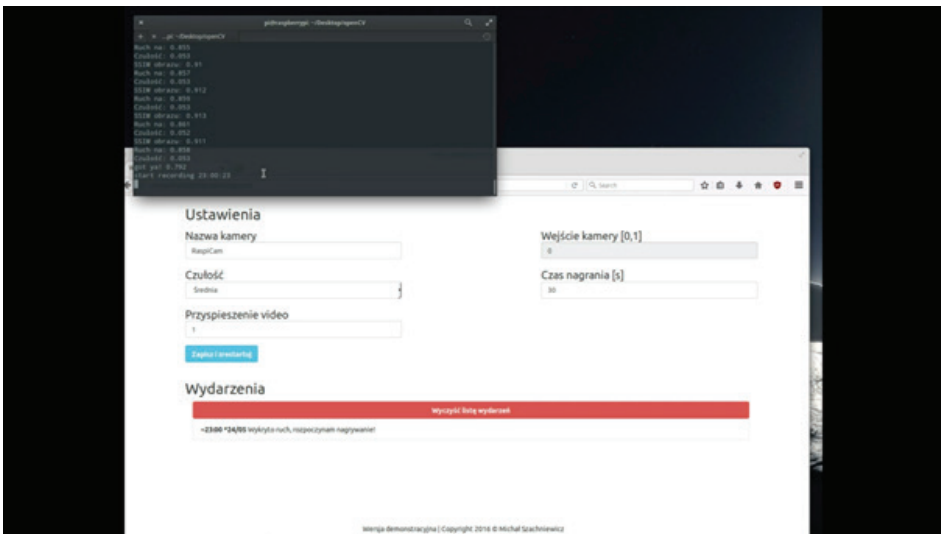
W celu potwierdzenia skuteczności aplikacji postanowiłem przeprowadzić eksperyment - załóżmy, że troskliwy ojciec chce dowiedzieć się, o której godzinie jego syn wróci do domu, jednak nie może długo czekać, ponieważ następnego dnia ma zawody wędkarskie i musi obudzić się już o świcie. Na szczęście przedtem owy syn pokazał mu nad czym pracował ostatnimi dniami, dzięki czemu ojciec może przygotować zasadzkę i położyć się spać. Zdarzenie zostało przedstawione na nagraniu z punktu 2. oraz na poniższych rysunkach.



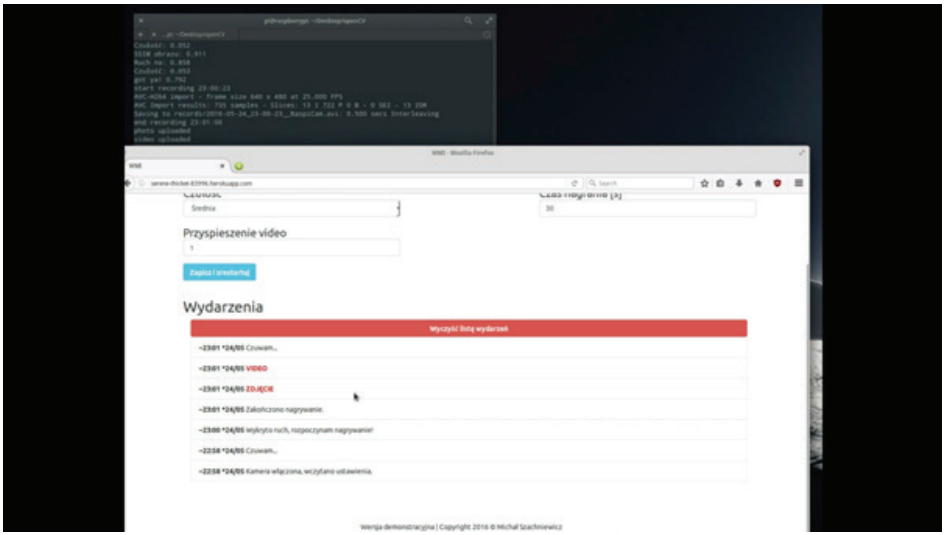
Rysunek 2 – stanowisko kamery podłączonej do RaspberryPi razem z diodami podczerwieni



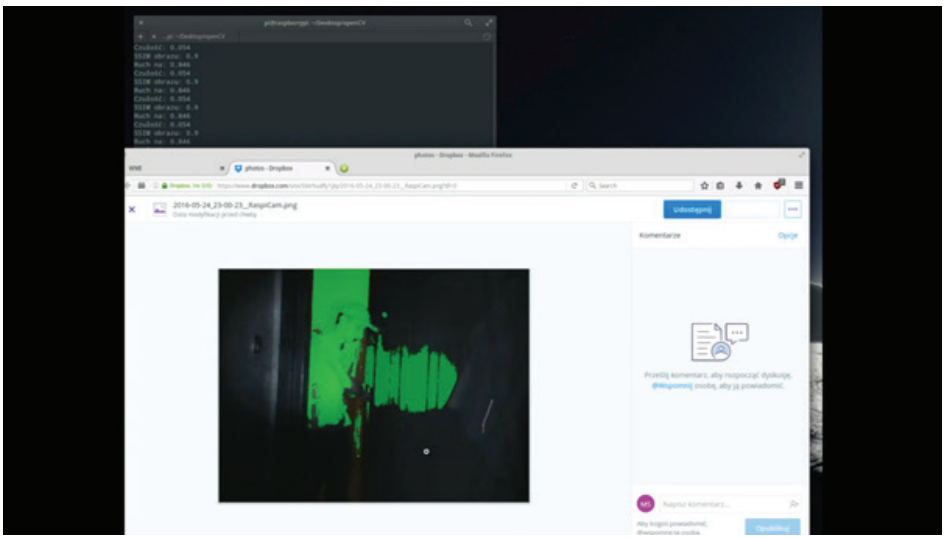
Rysunek 3 – nawiązywanie połączenia z RaspberryPi za pomocą SSH



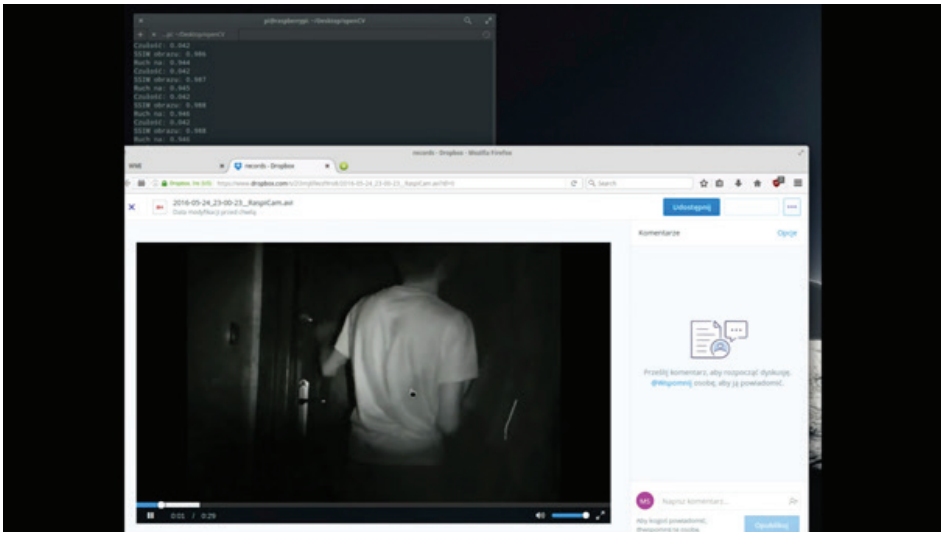
Rysunek 4 – widoczny w terminalu proces wykrywania ruchu



Rysunek 5 – pełen cykl komunikatów wysyłanych z kamery oraz linki do zdjęcia i video



Rysunek 6 – zdjęcie z zaznaczonym ruchem w momencie jego wykrycia



Rysunek 7 – video dokumentujące zdarzenie

Podsumowanie i dalsze prace rozwojowe

Aktualna wersja może się wydawać uboga, mimo to posiada już znaczną przewagę nad systemami monitoringu dostępnymi w sklepach - niski koszt. Wystarczy posiadać kamerkę USB i środowisko, na którym uruchomimy pythona z opencv - na przykład mikrokomputer Raspberry Pi. W porównaniu do średnich 500 zł za monitoring jest to kusząca oferta dla osoby, która chciałaby po prostu wiedzieć, co dzieje się w jej garażu, nawet gdy wyjeżdża z domu. Dla wymagających nietrudno byłoby dodać syrenę alarmową lub mailowe powiadomienia i autoryzację użytkowników.

Możliwe jest też nieco inne zastosowanie - jako foto-pułapki na zwierzęta. Oprócz znacznej redukcji kosztów takiej pułapki zyskujemy niewiarygodną czułość - kamera może reagować na zmianę nawet pojedynczego piksela (oczywiście lepiej ustawić słabszą czułość), a ponadto moduł OpenCV jest w stanie rozpoznawać poszczególne(wcześniej zaprogramowane) obiekty, dzięki

czemu właściciel kamery mógłby otrzymywać jedynie interesujące go materiały, na przykład dotyczące zwierząt określonego gatunku lub rozmiaru.Niewątpliwie byłoby to świetne rozwiązanie dla studentów biologii, leśników, czy hobbystów, aczkolwiek w tej chwili należy zadowolić się po prostu dodatkowym zabezpieczeniem majątku.

Bibliografia:

- "Nowoczesne aplikacje internetowe.", Jeff Dickey
- www.pyimagesearch.com
- www.pythonprogramming.net
- www.newthinktank.com

ELEKTRONICZNY SYSTEM ZARZĄDZANIA STOWARZYSZENIEM

Tomasz Dobrowolski
Akademickie Liceum Ogólnokształcące
Politechniki Wrocławskiej
tomasz.dobrowolski.2015@zsa.pwr.edu.pl

Wstęp

Zarządzanie jedną z polskich organizacji pozarządowych jest niezwykle trudne z uwagi na obowiązek prowadzenia księgowości, gromadzenia rozmaitej dokumentacji i składania rocznych sprawozdań finansowych, a także konieczność ciągłego kontaktu z członkami zarządzanej organizacji. Dość skomplikowany jest też obowiązujący w wielu stowarzyszeniach sposób zapisu nowych uczestników, który polega na wypełnieniu formularza zgłoszeniowego, sporządzeniu i wysłaniu podpisanej deklaracji członkowskiej, kilku oświadczeń oraz zainicjowaniu kontaktu z jednym z koordynatorów werbujących kandydatów. Tak złożona procedura często odstrasza osoby zainteresowane członkostwem, a zarządzających stowarzyszeniem zmusza do wielogodzinnej weryfikacji i korekty wypełnionych przez kandydatów formularzy oraz wprowadzania danych osobowych nowych członków do bazy danych. Na rynku jest dostępnych kilka programów znacznie ułatwiających prowadzenie organizacji pozarządowej, są to jednak bardzo rozbu-

dowane programy, których legalne użytkowanie wymaga wykupienia licencji, na którą znaczna część niewielkich stowarzyszeń nie jest w stanie sobie pozwolić. Wdrożenia takiego systemu nie ułatwia fakt, że większość organizacji nie posiada własnych komputerów, tylko korzysta z prywatnego sprzętu członków (przeważnie wieloosobowego) zarządu, a jedna licencja pozwala na zainstalowanie oprogramowania na jednym komputerze.

Cel pracy

Projektowany system ma pomagać w efektywnym prowadzeniu organizacji pozarządowej poprzez ułatwienie prowadzenia księgowości, umożliwienie szybkiego i skutecznego kontaktu z członkami stowarzyszenia oraz maksymalne uproszczenie procedury rejestracji kandydatów.

Technologie

System opiera się na relacyjnej bazie danych MySql, która jest obsługiwana przez serwer napisany w języku PHP; automatyczne generowanie dokumentów umożliwia biblioteka fpdf. Szatę graficzną stanowi wykonany w CSS i HTML szablon Spring Time autorstwa chocotemplates.

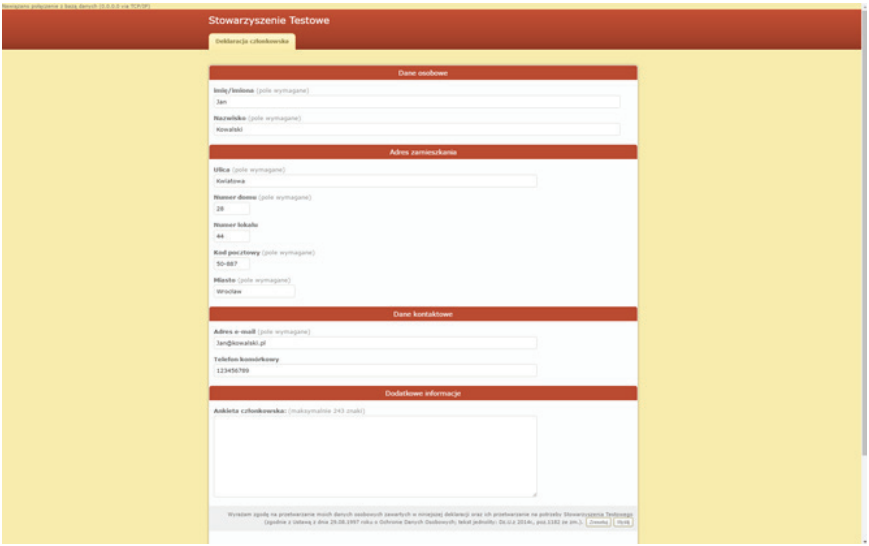
Opis funkcji systemu

Zgłoszenie nowego członka polega na wypełnieniu jednego formularza, co powoduje system automatyczne wpisanie użytkownika do bazy danych, listy mailingowej oraz (za pomocą biblioteki fpdf) wygenerowanie deklaracji członkowskiej w formacie .pdf. System umożliwia także bardzo skuteczny i wygodny kontakt z członkami organizacji (za pomocą automatycznego mailingu działającego dzięki pętli for i funkcji mail()) oraz ewidencjonowanie wpłacania składek członkowskich.

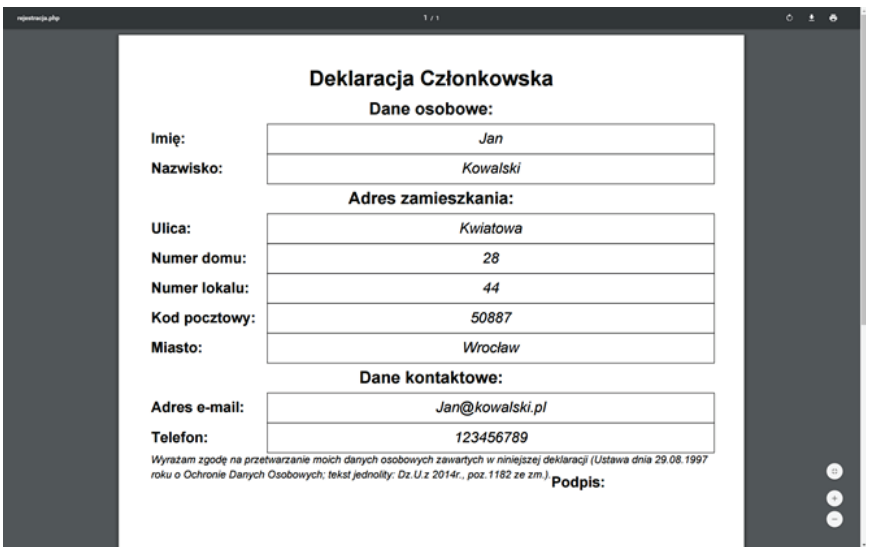
Plany na Przyszłość

Planowane jest wprowadzenia funkcji automatycznego generowania rocznych sprawozdań finansowych, powiadamiania członków organizacji poprzez sms oraz połączenie projektowanego systemu z jednym z systemów obsługujących mikropłatności, by ułatwić ewidencjonowanie opłaca

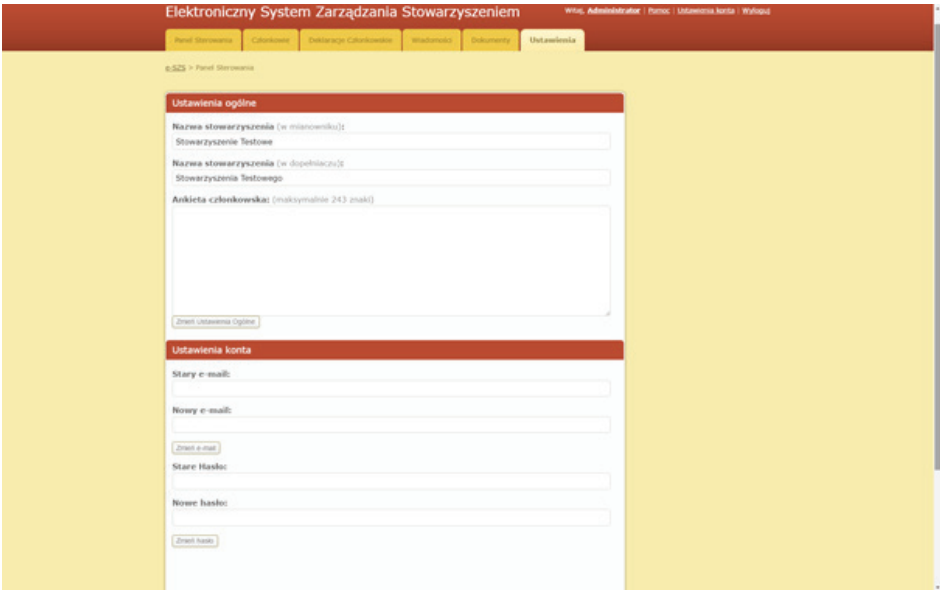
Galeria



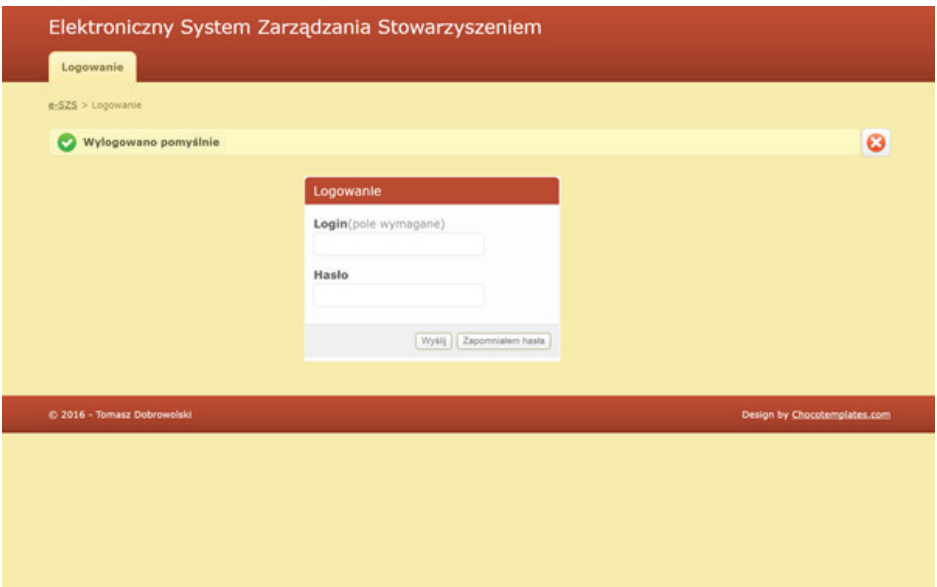
Rysunek 1- formularz rejestracyjny



Rysunek 2 – Wygenerowana deklaracja członkowska



Rysunek 3 – panel administratora - ustawienia



Rysunek 4 – panel administratora – logowanie

Fragment kodu 1- funkcja „zapomniałem hasła"

```
$logowanie = $db->query(„SELECT * FROM administratorzy WHERE login='$login'");
$tablica = $logowanie->fetch_assoc();
$znaki = array(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, q, w, e, r, t, y, u, i, o, p, a, s, d, f, g, h, j, k, l, z, x, c, v, b, n, m);          //Tablica znaków, spośród //których system generuje nowe hasło
for($i=0; $i < 8; $i++)              //Pętla losująca 8-znakowe hasło(pętla wykonuje się 8 razy)
{
$znak = ($znaki[array_rand($znaki)]);      //losowanie znaku
$nowehaslo = $nowehaslo.$znak;          //dopisywanie wylosowanego znaku do hasła składającego się ze znaków wylosowanych w //czasie poprzednich obrotów pętli
}
$nowehaslomd5=md5($nowehaslo);       //szyfrowanie wygenerowanego hasła za pomocą md5
$db->query(„UPDATE administratorzy set haslo='$nowehaslomd5' WHERE login='$login'");   //aktualizowanie bazy danych
mail($tablica[,email'], „E_SZS Nowe Hasło!","Witaj,.$login. Twoje nowe hasło to Elektronicznego Systemu Zarządzania Stowarzyszeniem to:.$nowehaslo. „, $od  = „From: Elektroniczny System Zarządzania Stowarzyszeniem \r\n");          //wysyłanie do użytkownika maila zawierającego nowe hasło
```

Bibliografia

http://chocotemplates.com/gui/spring-time/.
http://php.net/manual/en/index.php.
http://php.net/manual/en/function.mail.php
http://php.net/manual/en/function.array-rand.php
https://jacekk.info/articles/show/13
http://www.fpdf.org/
http://www.fpdf.org/en/tutorial/tuto1.htm

ROGUELAJK

Maciej Markuszewski, Kamil Bauer, Mariusz Mykicki

Akademickie Liceum Ogólnokształcące Politechniki Wrocławskiej

maciej.markuszewski.2014@zsa.pwr.edu.pl,
kamil.bauer.2014@zsa.pwr.edu.pl,
mariusz.mykicki.2014@zsa.pwr.edu.pl

I. Wstęp

Projekt powstał z inspiracji Kamila Bauera, który zajmuje się nim od strony graficznej. Wybraliśmy typ gry roguelike jako wielcy fani takich gier. Przykładami roguelike'ów są: Dwarf Fortress, The Binding Of Isaac, Rogue Legacy, Crypt Of The NecroDancer, Enter The Gungeon. Projekt spaja podstawowe założenia wszystkich tych gier. Program napisany został w p5, czyli bibliotece JavaScript studia Processing. W tworzeniu kodu pomagał Mariusz Mykicki. Gra jest umieszczona w sieci pod adresem http://kajtsweb.webd.pl/lolmaj/graj/ .

II. Założenia

W projekcie gry przyjęto następujące założenia realizacyjne:

- Gra odbywa się w lochu, który za każdym razem jest losowo generowany,
- By się poruszać gracz używa klawiszy „w" „s" „a" „d" lub klika myszką w wybranym kierunku,
- By ukończyć grę należy znaleźć schody, które są umieszczone w jednym z pokojów,
- W ukończeniu gry przeszkadzają przeciwnicy, którzy poruszają się według określonych algorytmów,

- Tury – po każdej akcji gracza, swoje ruchy wykonują przeciwnicy,
- Podczas rozgrywki gracz zdobywa przedmioty i zdolności pomagające mu w dalszej grze,
- Gracz nie widzi całego lochu, a tylko otoczenie swojej postaci w promieniu 5 kratek,
- Gracz zna kształt części lochu, którą już widział,
- Postęp uzyskany podczas rozgrywki nie jest zapisywany,
- Gracz odblokowuje nowe postacie spełniając odpowiednie warunki przejścia gry,
- Interfejs projektu opracowany jest w języku angielskim,
- Wszystkie grafiki wczytuje się do skryptu w czasie uruchamiania rozgrywki. Dzięki temu gracz nie potrzebuje ciągłego połączenia z Internetem. Dodatkowo ładowanie grafik nie spowalnia gry podczas rozgrywki.

III. Przykładowe algorytmy

W procesie projektowania gry opracowano różne algorytmy jej funkcjonowania. Do najważniejszych z nich można zaliczyć:

A. Dopasowanie rozmiarów gry do wielkości okna przeglądarki lub ekranu urządzenia mobilnego gracza:

1) Wykonuj za każdym razem gdy zmieniany jest rozmiar okna przeglądarki
2) Podziel okno przeglądarki na kwadraty o boku 32 pikseli,
3) Gra będzie zajmowała parzystą liczbę kwadratów wzdłuż osi X,
4) Gra będzie zajmowała nieparzystą liczbę kwadratów wzdłuż osi Y.

B. Generowanie za każdym razem innego lochu:

1) Środek lochu zdefiniuj jako pokój 1x1,
2) Losuj miejsce i wymiary nowego pokoju,
3) Jeśli ten pokój jest zbyt blisko innego tworząc jeden większy pokój, wróć do kroku drugiego,
4) Twórz korytarz w kierunku pokoju o numerze o jeden mniejszym,
5) Przerwij tworzenie korytarza, jeśli ten natknie się na istniejący pokój lub korytarz:

a) Dopóki masz tylko jedno pole przy końcu korytarza buduj dalej w danym kierunku.

6) Wypełnij wylosowany obszar pokoju polami,
7) Wylosuj, czy w pokoju będzie trawa lub woda,
8) Umieść na środku pokoju trawę lub wodę,
9) Wypełnij losowe pole pokoju wodą lub trawą. Jeśli jest to pole sąsiednie do istniejącej już wody lub trawy,

10) Sprawdź każde pole w pokoju z trawą, czy jest okrążone przez trawę. Jeśli tak, umieść tam jednego z dwóch grzybów,
11) Wylosuj, czy w pokoju będzie skrzynia. Jeśli tak, umieść ją na środku pokoju,
12) Po wybudowaniu wszystkich pokojów umieść przy bokach każdego z nich drzwi, jeśli się da:
a) Jeśli dane pole ma pola w osi poziomej, a w pionowej ściany lub na odwrót postaw drzwi.
13) Umieść w pokoju startowym grafikę startu, a w pierwszym pokoju schody.

C. Mgła wojny:
1) Wszystkie pola mapy mają wartość niewidzialne,
2) Po każdym ruchu sprawdź wszystkie pola w promieniu 5 kratek od postaci, czy linia łącząca środek pola z postacią ze środkiem danego pola jest przerwana przez ścianę lub drzwi. Jeśli nie to dane pole widać,
3) Jeśli dane pole nie jest widzialne w tym momencie, ale było go widać wcześniej, to ma wartość przysłonięte,
4) Wyświetl grafiki pól, które mają wartość widzialne lub przysłonięte,
5) Wyświetl grafiki ścian, które sąsiadują z co najmniej jednym polem o wartości widzialne lub przysłonięte,
6) Wyświetl grafiki potworów jeśli znajdują się na polu o wartości widzialne.

D. Efekt przebywania w wodzie:
1) Wyświetl grafiki wody na polach z wodą,
2) Wyświetl grafiki postaci, przedmiotów i potworów,
3) Wyświetl grafiki pasków wody na dole pól z wodą.

E. Wartości komórek w tablicy mapy:
1) Cyfry jedności odpowiadają za to, czy dana komórka jest ścianą, drzwiami, czy polem oraz w jakim stopniu je widać,
2) Cyfry dziesiątek i setek odpowiadają za to jaką dana komórka ma grafikę pola (start, schody, woda, trawa, grzyby),
3) Cyfry setek, dziesiątek i jedności tysięcy odpowiadają za to, czy dana komórka zawiera interakcję (skrzynię, monety, przedmioty),
4) Cyfry dziesiątek i jedności miliardów oraz setek dziesiątek i jedności milionów odpowiadają za postać (animacja, klasa, rasa),
5) Kolejne cyfry odpowiadają za potwory (animacja, id potwora).

F. Poruszanie się za pomocą myszy lub klawiatury ekranowej:
1) Podziel okno gry przekątnymi na cztery trójkąty,
2) Podziel okno gry liniami pod kątem 45 stopni do pionu/poziomu, które przecinają się na środku ekranu gry,
3) Jeśli gracz kliknie w część wspólną odpowiednich figur, to postać poruszy się w danym kierunku.

IV. Wiedza i umiejętności i nabyte w trakcie realizacji projektu

W trakcie realizacji projektu nabyto wiedzę dotyczącą samodzielnego uczenia się nowego języka programowania od podstaw.
Realizacja projektu umożliwiła również pozyskanie następujących umiejętności:
- Obsługa dużej liczby funkcji biblioteki p5,
- Poznanie podstaw JavaScript,
- Poznanie języka PHP do realizacji systemu rejestracji i logowania,
- Obsługa bazy danych MYSQL w celu realizacji systemu rejestracji i logowania.

V. Dalsze prace badawcze i rozwojowe

Prace związane z rozwojem projektu będą dotyczyły następujących zagadnień:
- System rejestracji i logowania:

Baza danych przechowywująca dane z formularza rejestracyjnego oraz skrypt w języku PHP sprawdzający zgodność loginu i hasła podczas logowania.
- Silnik:

Stworzenie skryptu, zawierającego główne mechaniki gry, którego nie trzeba będzie edytować by dodać nowe elementy projektu.
- Poradnik:

Stworzenie mapy uczącej gracza mechaniki i zasad gry.
- Przedmioty:

Ze skrzyń postać zdobywa przedmioty, które będą miały różne właściwości. Postać może mieć w danym momencie tylko jeden przedmiot danego typu w użytku. Zdobyte mikstury mają losowy, nieznany efekt dopóki się ich nie pozna. Przedmioty są podzielone na różne kategorie częstotliwości wystę-

powania. Informacje o przedmiotach będą zapisane w plikach o rozszerzeniu csv (comma-separated values).

• Opracowanie większej liczby postaci:

Stworzenie profesji i ras postaci, które będą się różniły początkowymi przedmiotami, umiejętnościami oraz statystykami. Nie wszystkie połączenia klasy z rasą będą możliwe.

• Umiejętności:

Czynności, które wymagają zużycia tury gracza.

• Kolejne poziomy lochów:

Po zakończeniu danego poziomu wejściem na schody generowany będzie nowy loch, który będzie charakteryzował się innym stylem graficznym oraz innymi potworami.

• Przeciwnicy:

Przeciwnicy będą losowo umieszczeni w wygenerowanym lochu. Ich sposób poruszania się będzie zależał od algorytmów im przypisanym. Przeciwnik może zmienić swój algorytm poruszania się na agresywny, gdy ujrzy postać gracza.

• System walki:

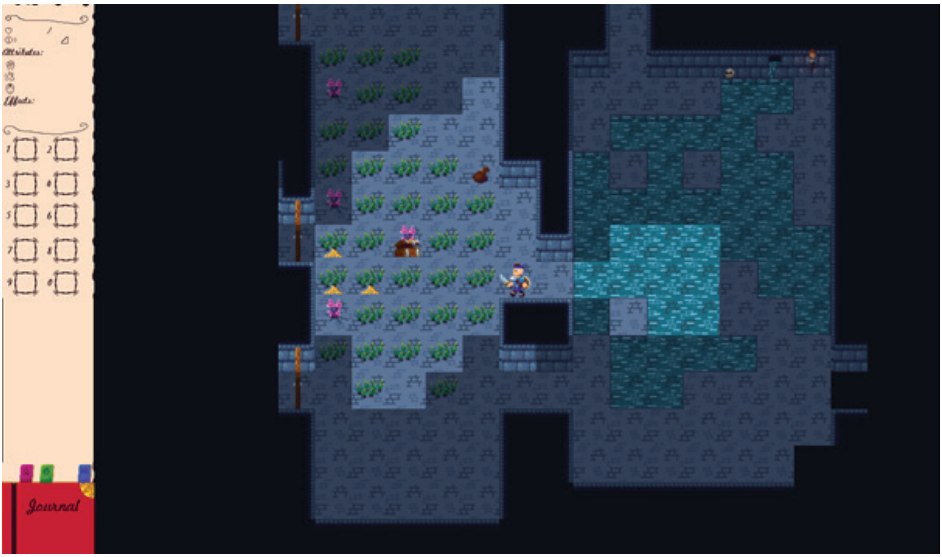
Gdy postać wykona ruch w na pole z przeciwnikiem lub przeciwnik wykona ruch na pole z postacią zadawany jest atak ujmujący celu punkty życia zgodne ze statystykami. Używanie niektórych umiejętności także może zadać obrażenia.

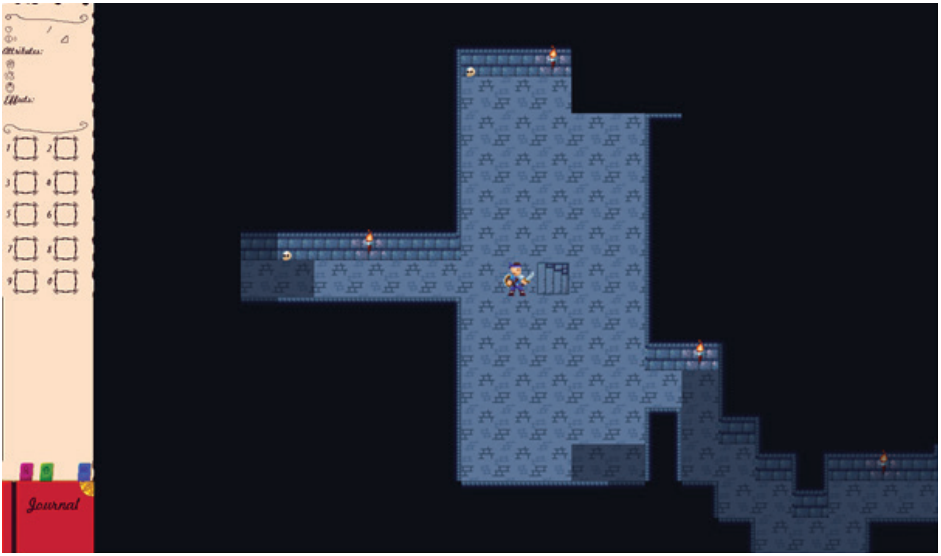
• Skróty klawiszowe:

Gracz będzie mógł przypisać klawiszom cyfr umiejętności, które będą się aktywowały po kliknięciu w dany klawisz. Po kliknięciu klawisza „f" gra będzie przechodziła w tryb pełnoekranowy. Po przytrzymaniu klawisza „r" przez 3 sekundy gra będzie się resetowała tworząc nową rozgrywkę. Gracz będzie mógł zmienić przypisanie wszystkich funkcji innym klawiszom.

VI. Podsumowanie

Projekt ten wykonano dla przyjemności, gdyż tworzenie gier oraz szeroko pojęte programowanie są wielką pasją autorów. Projekt zajmuje ponad 2000 linijek kodu oraz ponad 2MB pamięci, ale planuje się jego rozszerzenie oraz realizację innych projektów związanych z obszarem gier komputerowych.





VII. Źródła:

https://p5js.org/

http://miroslawzelent.pl/kurs-php/logowanie-do-strony-sesja-wstrzykiwanie-sql/

http://miroslawzelent.pl/kurs-php/rejestracja-captcha-hashowanie/