Politechnika Śląska w Gliwicach
Wydział Informatyki, Elektroniki i Informatyki

Programowania Komputerów

Dom Tower

autor	Dominik Florencki
prowadzący	dr inż. Roman Starosolski
rok akademicki	2016/2017
kierunek	informatyka
rodzaj studiów	SSI
semestr	4
termin laboratorium / ćwiczeń	Środa, 8:30 – 10:00
grupa	3
sekcja	2
termin oddania sprawozdania	2017-09-30
data oddania sprawozdania	2017-05-03

1 Treść zadania

Napisać jednoosobową grę platformową typu „icy tower”, w której gracz ma za zadanie skakać postacią w górę planszy. Gra ma posiadać kilka poziomów trudności, wiele utrudnień i bonusów.

2 Analiza zadania

Program jest grą jednoosobową, dlatego należy wziąć pod uwagę zarówno interakcję z użytkownikiem, jak i zagadnienia algorytmiczne zawarte w silniku gry. Gra posiada 5 poziomów trudności, zmieniające szerokość platform.

2.1 Struktury danych

W programie zostały listy ze standardowej biblioteki cpp std<list>

```
std::list<Floor *> floorList;
std::list<Bonus *> bonusList;
std::list<ToDraw *> interfaceList;
std::list<Player *> playerList;
std::list<Enemy *> enemyList;
```

Listy zawierają wskaźniki na odpowiednie elementy, klas bazowych, dzięki temu polimorfizm został efektywniej wykorzystany.

3 Specyfikacja wejściowa, bonusy i utrudnienia

W grze zostały zaimplementowane różnego rodzaju bonusy oraz utrudnienia:

W przypadku odpowiedniego rozpędzenia się gracza w poziomie, gracz otrzymuje zdolność do zrobienia SuperJump, o wiele większego niż normalny skok.

Bonusy:

 -bonus Duch, powoduje pojawienie się wroga gracza Ducha, który ściga gracza 

 dodaje + 3 punkty do konta gracza

 odejmuje -5 punktów z konta gracza

 obraca ekran rozgrywki

 teleportuje gracza na środek planszy

Rodzaje platform:

 normalna platforma, można się po niej poruszać w lewo i prawo, bez problemów

 drewniana platforma, nie można poruszać się w lewo, ani prawo, tylko skakać

 niebiańska platforma, wybija gracza na wysokość SuperJump

 tęczowa platforma, przesuwa się prawo-lewo, zarówno z graczem, jak i bez niego

 szara platforma, po pojawieniu się na niej gracza, spada w dół

4 Specyfika zewnętrzna-wewnętrzna

W programie, używane są dwa pliki nagłówkowe, gdzie są zadeklarowane wszystkie stałe. Są to pliki:

Header.h	// tutaj są wszystkie stałe używane w grze, takie jak wymiary okna, prędkość gracza, itp…
PathHeader.h	//tutaj są wszystkie ścieżki do plików z grafikami używanymi przez program

5 Specyfikacja wewnętrzna

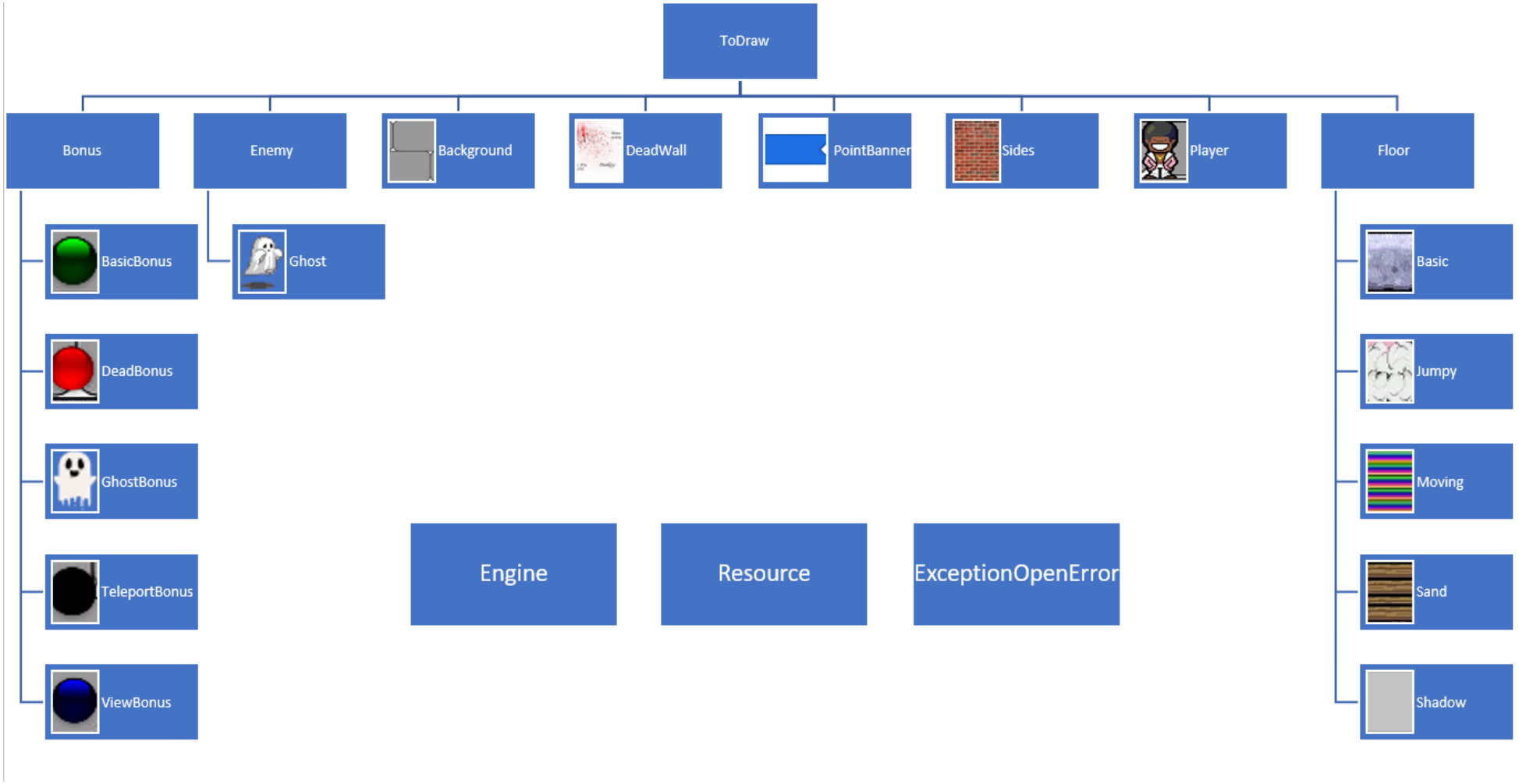
5.1 Ogólna struktura programu

Program generuje planszę oraz wszystkie jej elementy, większość elementów jest przechowywana w listach w celu usprawniania działania programu.

5.2 Podział programy na klasy

Program został podzielony na klasy, które wzajemnie po sobie dziedziczą. Dziedziczenie zostało pokazane na poniższym schemacie.

W dalszej części dokumentacji, bardziej znaczące zostaną dokładniej opisane.


ToDraw
Bonus
Enemy
Background
DeadWall
PointBanner
Sides
Player
Floor
BasicBonus
DeadBonus
GhostBonus
TeleportBonus
ViewBonus
Ghost
Basic
Jumpy
Moving
Sand
Shadow
Engine
Resource
ExceptionOpenError

5.3 Opis klas

5.3.1 Main

Klasa główna programu, tworzy obiekt engine, a następnie wywołuje włączenie gry.

5.3.2 Engine

Klasa zarządzająca całą rozgrywką z graczem. Przechowuje dane o przebiegu gry, punkty, listy elementów, stan gry, okno gry, klawisze gry.

```
std::list<Floor *> floorList;
std::list<Bonus *> bonusList;
std::list<ToDraw *> interfaceList;
std::list<Player *> playerList;
std::list<Enemy *> enemyList;
int lvl;
sf::View *view;
sf::RenderWindow *window;
sf::Vector2f gravity;
Resource *resource;
int points;
Game game;
Key key;
```

Zarządza główną pętlą gry, sprawdza, czy gracz przegrał, wygrał czy przechodzi na kolejny poziom gry.

```
void start();
```

5.3.2 Resource

Klasa zarządzająca wczytywaniem grafik do gry, a następnie udostępniania ich poszczególnym klasom. Zastosowanie wczytywania tylko jednej tej samej grafiki, zamiast podczas tworzenia każdego obiektu, spowodowało znaczące ograniczenia wykorzystania procesora.

```
Resource();
```

Konstruktor inicjuje wszystkie textury, wczytywane z pliku nagłówkowego PathHeader.h.

Do inicjalizacji służą metody init…()

A do nadania sprite danej textury, motody set…()

5.3.3 ExceptionOpenError

Klasa, która jest wykorzystywana jako wyrzucony wyjątek, zawiera:

```
std::string messege;
int code;
```

5.3.4 ToDraw

Klasa Bazowa większości klas programu, posiada tylko klasy virtualne.

5.3.5 Bonus

Klasa odpowiedzialna za różne bonusy w grze, posiada prywatnego sf::sprite i posiada metody, które są dziedziczone przez klasy pochodne.

5.3.6 Enemy

Klasa bardzo podobna do klasy Bonus, oprócz sprite, posiada jeszcze informacje o kierunku poruszania się przeciwnika.

5.3.7 Klasy Background, Deadwall, PointBanner, Sides

Klasy odpowiedzialne za interface gry, są odpowiedzialne za kolejne wyświetlanie i poruszanie tła, tła końca gry, liczy punktów, i boków planszy.

5.3.8 Player

Klasa gracza, oprócz takich podstawowych metod, jak metody odpowiedzialne za poruszanie się i zwalnianie postaci, są również metody sprawdzające kolizje z innymi elementami.

5.3.9 Floor

Klasa Bazowa wszystkich rodzajów platform, przechowuje sf::sprite, i posiada metody powiązane z obsługą właśnie tego sprite.

5.4 Opis działania programu

Klasa Engine tworzy nowe obiekty sf::RenderWindow i Resource i inicjalizuje wszystkie elementy planszy, metodami,

```
initResource();

setBoard();
```

W metodzie start() jest główna pętla gry, gdzie kolejno wykonywane są metody:

```
while (game == newGame) {
	setBoard();
	while (window->isOpen() && (game < 2)) {
		sf::Event event;
		while (window->pollEvent(event))
		{
			if (event.type == sf::Event::Closed)
				window->close();
		}
		if (game == onGoing) {
			newElements();			//dodanie nowych elementow
			enemy();			//wykonywanie ruchu enemy
			player();			//wykonywanie ruchu player
			moveFloor();			//wykonywanie ruchu platform
		}
		checkGame();				//sprawdenie stanu gry
		draw();					//wyswietlenie na ekranie
	}
	deleteAll();					//usuwanie wszystkich elementow
	initParam();					//wartosci poczatkowe
}
```

6 Zastosowane zagadnienia wyższego programowania C++

6.1 Wyjątki

Wyjątki zostały zastosowane do obsługi grafik w sytuacji wyjątkowej, czyli gdy nastąpił błąd wczytania w klasie Resource:

```
void Resource::initPlayer()
{
	if (!playerIm.loadFromFile(PATH_PLAYER)) {
		throw new ExceptionOpenError(PATH_PLAYER,2);
	}
	sf::Color color = playerIm.getPixel(5, 5);
	playerIm.createMaskFromColor(color, 0);
	if (!playerTex.loadFromImage(playerIm)) {
		throw new ExceptionOpenError(PATH_PLAYER, 3);
	}
}
```

Które, następnie są wyłapane w klasie Engine z zastosowaniem odpowiedniej reakcji programu.

```
void Engine::initResource()
{
	try {
		resource = new Resource();
	}
	catch (ExceptionOpenError *e) {
		std::cout << e->getString() << e->getCode() << std::endl;
		game = close;
	}
}
```

6.2 Dynamic_cast

Dynamic_cast został zastosowany w celu rozpoznania klasy obiektu ze wskaźnika klasy bazowej:

```
		//platforma poruszajaca sie
			if (Moving *floorMoving = dynamic_cast<Moving*> (floor)) {
				player->makeMove(floorMoving->getMove());
			}
```

6.3 Kontenery std::list

W celu przechowywania dynamicznego danych w programie, posłużyłem się standardowymi kontenerami typu listy

```
	std::list<Floor *> floorList;
	std::list<Bonus *> bonusList;
	std::list<ToDraw *> interfaceList;
	std::list<Player *> playerList;
	std::list<Enemy *> enemyList;
```

6.4 Iteratory

W celu poruszania się i kasowania elementów listy potrzebowałem użyć iteratorów:

```
	for (std::list<Player*>::iterator playerIt = playerList.begin(); playerIt != 
playerList.end(); playerIt++) {
		if (*playerIt == player) {
			delete *playerIt;
			playerList.erase(playerIt);
			return;
		}
```

```
}
```

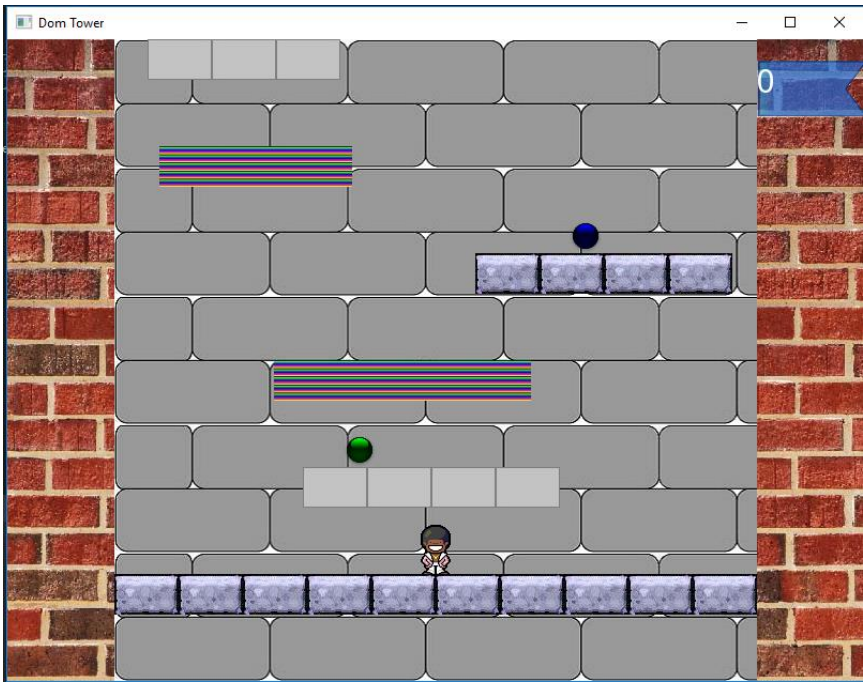
6.5 Algorithm

W różnych zastosowaniach, jak na przykład:

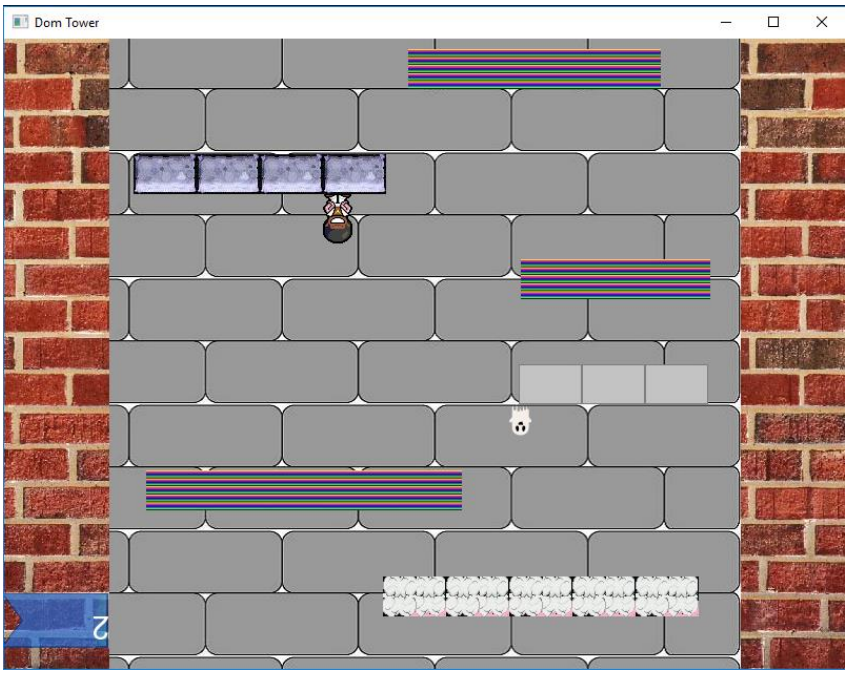
```
std::swap(key.left, key.right);
```

7 Testowanie

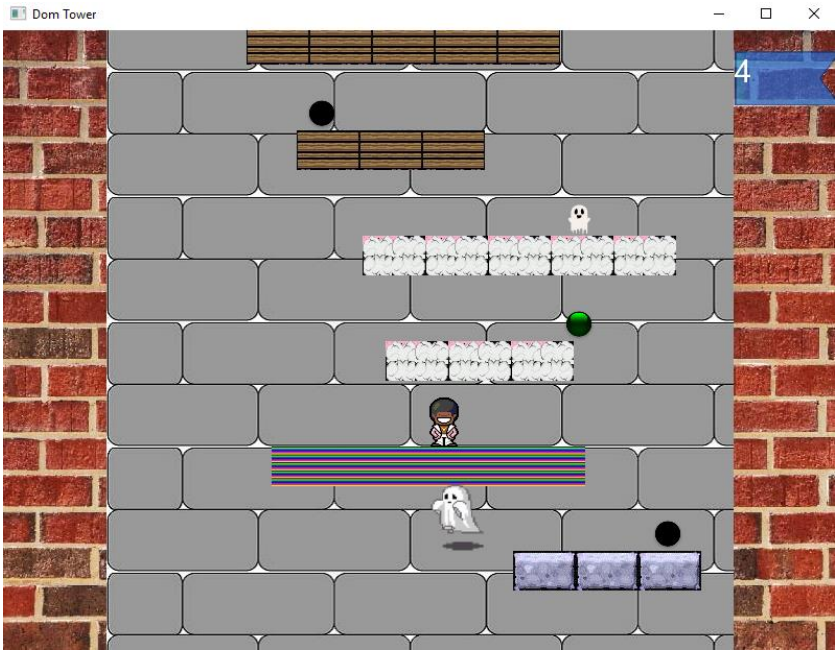
Program został przetestowany i działa poprawnie, został również przetestowany w związku z wyciekami pamięci.



Po zebraniu niebieskiego bonusa:



Po zebraniu bonusa Duch



Kolejny level gry, platformy się zmniejszają

6 Wnioski

Moim głównym zadaniem było odtworzenie i w znacznym stopniu poprawienie jej możliwości, przez dodanie autorskich pomysłów, po rozmowie z prowadzącym, popularnej gry w języku C++, używając myślenia obiektowego, używając możliwości wyższego poziomu, poznanych najpierw na wykładach, a później przećwiczonych na laboratoriach oraz implementując to w kodzie. Zadanie nie należało do najprostszych, zarówno analiza strukturalna pliku, jak i podział na klasy, było wymagającym zadaniem. Niemniej jednak, dzięki poznanym na laboratoriach zasadach udało mi się stworzyć grę. Dzięki obiektowości, grę można łatwo rozbudować, o nowe poziomy, noywch przeciwników, czy nowe bonusy oraz zmieniać jej parametry, zmieniając tylko kilka linii kodu, oczywiście kod też jest bardziej przejrzysty dzięki dużej liczbie klas, ale przede wszystkim, dzięki zastosowaniu standardów C++ co przyczyniło się do znacznego ograniczenia błędów podczas tworzenia jak i do szybkiego ich znalezienia podczas jego testowania.