**Digger**

Курсов проект по ООП-практикум,

ФМИ, СУ, летен семестър 2020/2021

### Описание на проекта

Проектът реализира графично приложение, наподобяващо всеизвестната игра Digger направена 1983 година. Дава възможност да се играе играта на по-ново устройство тъй като оригиналната игра е направена 1983 година и не може да се възпроизведе на модерен хардуер.

### Как се играе

Като се стартира играта първо излиза главното меню и на него се показват десетте най-високи резултата постигнати на този компютър. За да се стартира играта е необходимо да се натисне някоя от стрелките или ” space bar”. След като играта е стартирана със стрелките се движим, а с ” space bar” изстрелваме куршум в посоката в която гледаме. За да стреляме отново е необходимо да изчакаме. Куршума ако уцели врага(които се контролира от компютъра) го унищожава и получава точки(250 точки). Също така играча събира емералди като ги докосва и получава точки за тях(25 точки). По картата има и торби с пари които може да се местят наляво и надясно от играча, когато се подкопае под тях те падат. Ако паднат от повече от 2 блока те се чупят и на тяхно място се показват пари. Тези пари могат да бъдат събрани от играча (с докосване от играча) той получава точки (250 точки). Торбата с пари ако докосне играч или враг той умира. Играча умре ли губи живот ако загуби всички животи играта приключва и се връща в главното меню. След определено време на картата се появява бонус(черешка) която при докосване носи 1000 точки. Също така дава възможността на играча да изяжда враговете си при докосване като получава 200 точки за всеки изяден враг.

### Архитектура

AnimatedSprite

Класа е взет от интернет и е модифициран. Класа се грижи за анимациите които се показват на екрана. За кадрите кой след кой е и дали да се повтаря анимацията.

void update(sf::Time deltaTime) – обновява анимацията спрямо времето което е минало между два кадъра.

void play() – стартира анимация която вече е била спряна веднъж.

void play(const Animation& animation) – стартира новата анимация.

void pause() – спира анимацията на последния кадър.

void stop() – спира анимацията и я връща на първия кадър.

void setLooped(bool looped) – задава дали да се повтаря анимацията или да спре на последния кадър.

Animation

Класа е взет от интернет и е модифициран. Класа съдържа анимация която може да бъде възпроизведена от AnimatedSprite.

void addFrame(sf::IntRect rect) – добавя нов кадър който е част от текстурата която е дадена предварително.

void setSpriteSheet(const sf::Texture& texture) – задаване на текстура от която да се взимат части за анимациите.

Bonus

Класа представлява бонуса който се появява на картата.

void render(sf::RenderTarget\* target) – показва бонуса на определения RenderTarget(в този случай е прозореца на играта).

Bullet

Класа представлява куршума който се изстрелва от играча и се самоунищожава при докосване на стена на тунела.

void update(sf::Time& elapsedTime, Map\* map) – обновява куршума и проверява дали е ударил стена на подадената карта.

void render(sf::RenderTarget\* target) - показва куршума на определения RenderTarget(в този случай е прозореца на играта).

DiggerPlayer

Класа представлява играча в него е цялата логика за движение и стреляне.

void update(sf::Time& elapsedTime, Map\* map) – проверява дали да премести играча и дали да прокопае нов тунел на картата.

void render(sf::RenderTarget\* target) - показва играча на определения RenderTarget(в този случай е прозореца на играта).

void deleteBullet() – унищожава куршума(използва се когато куршума докосне враг).

void deathPlayer() – сменя анимацията на играча на надгробна плоча.

void setPosition(int x,int y) – задава нова позиция на играча.

Emerald

Класа представлява емералд които може да бъде събиран.

void render(sf::RenderTarget\* target) - показва емералда на определения RenderTarget(в този случай е прозореца на играта).

Enemy

Класа представлява противника. Може да си сменя типа (Hobbin, Nobbin). Движението на противника се променя в зависимост какъв му е моментния тип.

void update(sf::Time& elapsedTime, Map\* map , DiggerPlayer\* player) – обновява анимацията на къде да се движи, дали да копае тунели в картата, или да изяжда емералди, торби или играча.

void render(sf::RenderTarget\* target) - показва емералда на определения RenderTarget(в този случай е прозореца на играта).

Game

Класа съхранява всички ресурси и това позволява да се предават референции което решава проблема с копирането на големите ресурси и подобрява кадрите в секунда.

void run() – стартира играта.

HighScoreSystem

Класа се грижи за зареждането и записването на постиженията(рекордите).

void updateHightScore() – обновява подредбата на резултатите от голям към малък.

void reseteCurrentScore() – занулява текущия резултат.

void addToCurrentScore(int points) – добавя определен брой точки към сегашния резултат.

MainMenu

Класа показва десетте най високи резултата и е междинен между две игри.

void update(sf::Time& elapsedTime) – обновява анимациите и проверява дали е натисната някоя от стрелките или ” space bar”

void render(sf::RenderTarget\* target) - показва анимациите и текстовете на определения RenderTarget(в този случай е прозореца на играта).

Map

Картата съдържа цялата информация къде са тунелите, емералдите, стартовата позиция на играча и на враговете.

void update(sf::Time& elapsedTime) – обновява торбите с пари дали трябва да се движат или падат.

void render(sf::RenderTarget\* target) - показва торбите с пари, емералдите, бонуса, фона и тунелите на определения RenderTarget(в този случай е прозореца на играта).

void createCircle(int x, int y) – създава кръг на подадената позиция ако няма такъв.

void spawnBonus() – създава бонуса ако вече не съществува.

void deleteBonus() – изтрива бонуса ако съществува такъв.

MoneyBag

Класа съдържа логиката за торбите с пари(Движението им наляво надясно и падането когато е подкопано под тях).

void update(sf::Time& elapsedTime, Map\* diggedSpots) – обновява торбата дали да пада, да се движи наляво или надясно.

void moveLeft() – торбата ще се придвижи наляво.

void moveRight() - торбата ще се придвижи надясно.

Play

Класа се грижи за кога два обекта се докоснат и какво да става с тях.

void update(sf::Time& elapsedTime) – обновява карта, играча, враговете, дали два обекта са се сблъскали и всичко което се случва в играта.

ResourceHolder

Класа е взет от интернет и е модифициран. Това е темплейтен клас които улеснява съхраняването и извличането на текстури, анимации и други.

void load(Identifier id, const std::string& filename) – зарежда ресурса от файл.

void insertResource(Identifier id, std::unique\_ptr<Resource> resource) – добавя вече зареден в паметта обект.

Resource& get(Identifier id) – взима ресурсите.

ResourceIdentifiers

Класа е взет от интернет и е модифициран. В него се декларира темплейтения клас ResourceHolder и допълнително улеснява работата с ресурси(Анимации, текстури и други).

Score

Класа съдържа само един резултат и е използван в HighScoreSystem. Веднъж създаден обекта не може да се променя единствено да се вижда какви са стойностите.

StringHelpers

Класа позволява от различен тип да се превърне в стрин.

std::string toString(const T& value) – превръща различен тип променлива в стринг.

### Обобщение на реализацията

Реализацията покрива по-голямата част от базовите изисквания на проекта.

### Възможности за разширение

Една възможност за допълнителна функционалност добавянето на възможност много хора да играят едновременно.

### Заключение

Проектът е реалиализиран успешно, спайзвайки основните ООП принципи.

### Използвана литература и външни препратки

<http://www.futrega.org/digger/>

<https://en.wikipedia.org/wiki/Digger_(video_game)#Music>

<https://oiipdf.com/sfml-game-development>