НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ «КПІ»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра програмного забезпечення комп’ютерних систем

**КУРСОВА РОБОТА**

***з дисципліни "Основи програмування"***

Виконав: Сорочинський Владислав Володимирович

Група: КП-52

Допущено до захисту

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

2 семестр 2015/2016

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ «КПІ»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра програмного забезпечення комп’ютерних систем

|  |  |
| --- | --- |
| Узгоджено    Керівник роботи    \_\_\_\_\_\_\_\_\_\_/Гадиняк Р.А./ | ЗАХИЩЕНА "\_\_"\_\_\_\_\_\_\_\_\_2016р.  з оцінкою\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_/Гадиняк Р.А./ |

Маріо

Виконавець роботи

Сорочинський Владислав Володимирович

\_\_\_\_\_\_\_\_\_\_\_\_\_\_2016р.

**Опис системи**

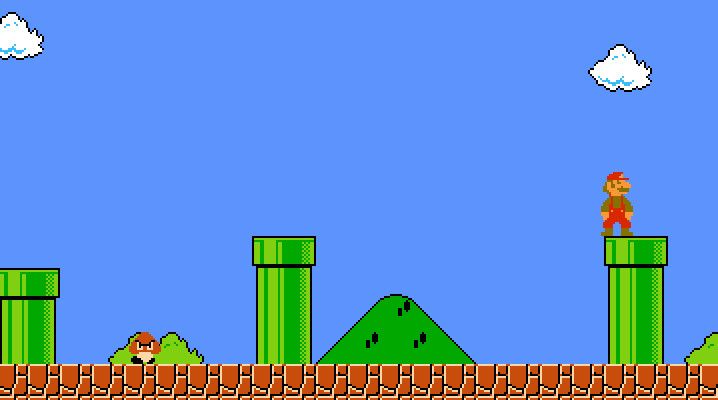
Програма являє собою 2D платформер у системі координат ХУ. Персонаж біжить по землі, стрибає по платформах, збирає монетки та стрибає згори на ворогів, убиваючи цим їх. Персонаж може померти, якщо неправильно стрибне на ворога або підійде до нього збоку.

Програма може використовуватися у розважальних цілях.

**Існуючі аналоги**

**Super Mario**

Головний герой саннтехнік Маріо змушений пройти купу випробувань, рівнів, ворогів, щоб бути разом зі своєю коханою.



[**Sonic the Hedgehog**](http://www.solarsystemscope.com/ru)

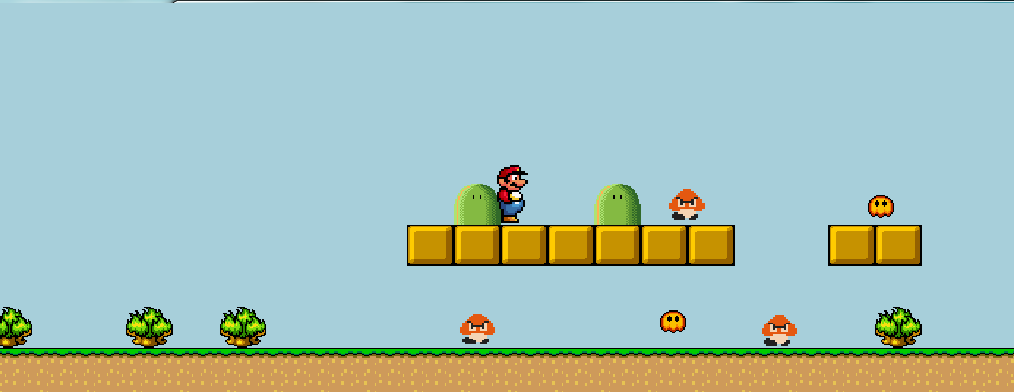
Головний герой – синій їжак Сонік, який бігая по рівням, перемагаючи своїх ворогів. Його мета – перемогти свого найбільшого ворога доктора Роботника.

****

**Основний функціонал системи**

Програма має наступні можливості та особливості:

* Гравітація, персонаж може бігти вліво, вправо та стрибати
* При досяганні певної точки він переміщається на наступний рівень
* Герой має 3 житті, які може втратити, впавши в ущелину або бути убитим ворогом
* Камера, що слідкує за героєм
* 3 рівні
* Можливість збирати монетки
* Платформи, що рухаються
* 3 види ворогів: легкі, середні, складні.
* Меню та екран перемоги
* Звуки

**Скетчі графічного інтерфейсу**

**Опис модулів програми**

Entity – батьківський клас для героя, ворогів та інших різноманітних об’єктів.

Enemy, Player, Fire, Coin, MovingPlatform – нащадки Entity.

|  |
| --- |
| entity.h |
| #ifndef ENTITY\_H\_INCLUDED  #define ENTITY\_H\_INCLUDED  #include <SFML/Graphics.hpp>  #include <string>  #include <vector>  #include <map>  #include <SFML/Graphics.hpp>  #include <iostream>  #include "level.h"  #include "view.h"  #include "tinyxml.h"  #include "tinystr.h"  using namespace sf;  using namespace std;  class Entity {  public:  std :: vector <Object> obj;  float dx, dy, x, y, speed, moveTimer, CurrentFrame;  int width, height, lives;  bool onGround, life, isMove;  Texture texture;  Sprite sprite;  String name;  Entity(Image &image, String Name, float X, float Y, int W, int H){  x = X;  y = Y;  width = W;  height = H;  name = Name;  moveTimer = 0;  CurrentFrame = 0;  speed = 0;  dx = 0;  dy = 0;  onGround = false;  isMove = false;  life = true;  texture.loadFromImage(image);  sprite.setTexture(texture);  sprite.setOrigin(width / 2, height / 2);  }  virtual void update(float time) = 0;  FloatRect getRect(){  return FloatRect(x, y, width, height);  }  };  class Player : public Entity {  public:  enum { left, right, up, jump, stay } state;  int Score;  float CurrentFrame;  SoundBuffer jumpBuffer;  Sound jumpSound;  Player(Image &image, String Name, Level &lev, float X, float Y, int W, int H) :Entity(image, Name, X, Y, W, H ){  CurrentFrame = 0;  Score = 0;  state = stay;  onGround = false;  lives = 3;  obj = lev.GetAllObjects();  jumpBuffer.loadFromFile("sounds/jump.wav");  jumpSound.setBuffer(jumpBuffer);  if (name == "Player"){  sprite.setTextureRect(IntRect(5, 6, 16, 38));  sprite.setScale(1.5, 1.5);  }  }  void control(){  if (Keyboard::isKeyPressed){  if (Keyboard::isKeyPressed(Keyboard::Left)) {  state = left;  speed = 0.1;  }  if (Keyboard::isKeyPressed(Keyboard::Right)) {  state = right;  speed = 0.1;  }  if ((Keyboard::isKeyPressed(Keyboard::Up)) && (onGround)) {  state = jump;  dy = -0.6;  onGround = false;  jumpSound.play();  }  }  }  void checkCollisionWithMap(float Dx, float Dy) {  for (int i = 0; i<obj.size(); i++)  if (getRect().intersects(obj[i].rect)) {  if (obj[i].name == "solid") {  if (Dy>0) {  y = obj[i].rect.top - height;  dy = 0;  onGround = true;  }  if (Dy<0) {  y = obj[i].rect.top + obj[i].rect.height;  dy = 0;  }  if (Dx>0) {  x = obj[i].rect.left - width;  }  if (Dx<0) {  x = obj[i].rect.left + obj[i].rect.width;  }  }  }  }  void update(float time)  {  control();  switch (state)  {  case right:  dx = speed;  if (speed != 0) {  if (CurrentFrame > 4) CurrentFrame = 0;  switch (int(CurrentFrame)) {  case 0:  sprite.setTextureRect(IntRect(5, 6, 14, 30));  break;  case 1:  sprite.setTextureRect(IntRect(21, 6, 16, 30));  break;  case 2:  sprite.setTextureRect(IntRect(39, 7, 16, 29));  break;  case 3:  sprite.setTextureRect(IntRect(57, 6, 16, 30));  break;  }  }  else  sprite.setTextureRect(IntRect(5, 6, 14, 30));  break;  case left:  dx = -speed;  if (speed != 0) {  if (CurrentFrame > 4) CurrentFrame = 0;  switch (int(CurrentFrame)) {  case 0:  sprite.setTextureRect(IntRect(19, 6, -14, 30));  break;  case 1:  sprite.setTextureRect(IntRect(37, 6, -16, 30));  break;  case 2:  sprite.setTextureRect(IntRect(55, 7, -16, 29));  break;  case 3:  sprite.setTextureRect(IntRect(73, 6, -16, 30));  break;  }  }  else  sprite.setTextureRect(IntRect(21, 6, -17, 39));  break;  case up:  break;  case stay:  sprite.setTextureRect(IntRect(5, 6, 14, 30));  break;  }  CurrentFrame += 0.005 \* time;  x += dx\*time;  checkCollisionWithMap(dx, 0);  y += dy\*time;  checkCollisionWithMap(0, dy);  sprite.setPosition(x + width / 2, y + height / 2);  if (lives == 0){  life = false;  }  if (!isMove){  speed = 0;  }  setPlayerCoordinateForView(x, y);  if (life) {  setPlayerCoordinateForView(x, y);  }  dy = dy + 0.0015\*time;  }  };  class Enemy : public Entity{  public:  float deathTimer;  Enemy(Image &image, String Name,Level &lvl, float X, float Y, int W, int H) :Entity(image, Name, X, Y, W, H){  obj = lvl.GetAllObjects();  deathTimer = 0;  if (name == "EasyEnemy"){  lives = 1;  sprite.setTextureRect(IntRect(0, 4, 16, 16));  dx = 0;  sprite.setScale(2, 2);  }  if (name == "MediumEnemy") {  lives = 2;  sprite.setTextureRect(IntRect(61, 1, 16, 23));  dx = 0.075;  sprite.setScale (2, 2);  }  if (name == "HardEnemy"){  lives = 3;  dx = 0.1;  sprite.setScale(2.5, 2.5);  }  }  void checkCollisionWithMap(float Dx, float Dy)  {  for (int i = 0; i < obj.size(); i++)  if (getRect().intersects(obj[i].rect))  {  if (obj[i].name == "solid" || obj[i].name == "stop"){//åñëè âñòðåòèëè ïðåïÿòñòâèå (îáúåêò ñ èìåíåì solid)  if (Dy>0) { y = obj[i].rect.top - height; dy = 0; onGround = true; }  if (Dy<0) { y = obj[i].rect.top + obj[i].rect.height; dy = 0; }  if (Dx>0) {  x = obj[i].rect.left - width;  if (name == "MediumEnemy")  dx = -0.075;  if (name == "HardEnemy")  dx = -0.1;  sprite.scale(-1, 1); }  if (Dx<0) {  x = obj[i].rect.left + obj[i].rect.width;  if (name == "MediumEnemy")  dx = 0.075;  if (name == "HardEnemy")  dx = 0.1;  sprite.scale(-1, 1); }  }  }  }  void update(float time)  {  CurrentFrame += 0.002 \* time;  if (lives == 0) {  deathTimer += 0.005 \* time;  height = 35;  }  if (deathTimer > 1)  life = false;  if (name == "EasyEnemy"){  if (CurrentFrame > 2)  CurrentFrame = 0;  if (lives != 0) {  switch ((int) CurrentFrame) {  case 0:  sprite.setTextureRect(IntRect(0, 4, 16, 16));  break;  case 1:  sprite.setTextureRect(IntRect(30, 4, 16, 16));  break;  }  }  }  if (name == "MediumEnemy") {  if (lives == 0)  life = false;  moveTimer += time;  if (moveTimer > 5000){  dx \*= -1;  sprite.scale(-1, 1);  moveTimer = 0;  }  if (CurrentFrame > 2)  CurrentFrame = 0;  if (lives != 0) {  switch ((int) CurrentFrame) {  case 0:  sprite.setTextureRect(IntRect(61, 1, 16, 23));  break;  case 1:  sprite.setTextureRect(IntRect(91, 1, 16, 24));  break;  }  }  }  if (name == "HardEnemy") {  if (lives == 0)  life = false;  moveTimer += time;  if (moveTimer > 6000){  dx \*= -1;  sprite.scale(-1, 1);  moveTimer = 0;  }  if (CurrentFrame > 4)  CurrentFrame = 0;  if (lives != 0) {  switch ((int) CurrentFrame) {  case 0:  sprite.setTextureRect(IntRect(162, 2, 31, 31));  break;  case 1:  sprite.setTextureRect(IntRect(201, 2, 32, 30));  break;  case 2:  sprite.setTextureRect(IntRect(242, 2, 31, 31));  break;  case 3:  sprite.setTextureRect(IntRect(281, 2, 32, 31));  break;  }  }  }  x += dx\*time;  checkCollisionWithMap(dx, 0);  sprite.setPosition(x + width, y + height);  }  };  class Coin : public Entity {  public:  Coin (Image &image, String Name,Level &lvl, float X, float Y, int W, int H) :Entity(image, Name, X, Y, W, H){  obj = lvl.GetObjects("solid");  sprite.setTextureRect(IntRect(7, 0, 18, 18));  }  void update (float time) {  sprite.setPosition(x + 10, y + 10);  }  };  class Exit : public Entity {  public:  Exit (Image &image, String Name,Level &lvl, float X, float Y, int W, int H) :Entity(image, Name, X, Y, W, H){  //obj = lvl.GetObjects("solid");  sprite.setTextureRect(IntRect(7, 0, 1, 1 ));  }  void update (float time) {  sprite.setPosition(x + 10, y + 10);  }  };  class Fire : public Entity {  public:  Fire (Image &image, String Name,Level &lvl, float X, float Y, int W, int H) :Entity(image, Name, X, Y, W, H){  //obj = lvl.GetObjects("solid");  sprite.setTextureRect(IntRect(7, 0, 1, 1));  }  void update (float time) {  sprite.setPosition(x + 10, y + 10);  }  };  class MovingPlatform : public Entity{  public:  MovingPlatform(Image &image, String Name, Level &lvl, float X, float Y, int W, int H) :Entity(image, Name, X, Y, W, H){  sprite.setTextureRect(IntRect(0, 0, W, H));  dx = 0.08;  }  void update(float time).  {  x += dx \* time;  moveTimer += time;  if (moveTimer>2000) { dx\*= -1; moveTimer = 0; sprite.setPosition(x+width/2, y+height/2); }  };  #endif // ENTITY\_H\_INCLUDED |

Реалізація: https://github.com/SorochynskyiVladyslav/CoursesRepository/blob/master/courses/prog\_base\_3/project%201.0/entity.h

View – модуль для керування камерою.

|  |
| --- |
| view.h |
| #ifndef VIEW\_H\_INCLUDED  #define VIEW\_H\_INCLUDED  #include <SFML/Graphics.hpp>  using namespace sf;  View view;  void setPlayerCoordinateForView(float x,float y) {  float tempX = x;  float tempY = y;  if (y > 600) tempY = 600;  if (y < 300) tempY = 300;  if (x < 400) tempX = 400;  view.setCenter(tempX, tempY);  }  #endif // VIEW\_H\_INCLUDED |

Реалізація: https://github.com/SorochynskyiVladyslav/CoursesRepository/blob/master/courses/prog\_base\_3/project%201.0/view.h

Level – допоміжний модуль для обробки рівня в форматі .tmx, що записає дані в форматі xml.

|  |
| --- |
| level.h |
| #ifndef LEVEL\_H\_INCLUDED  #define LEVEL\_H\_INCLUDED  #include <string>  #include <vector>  #include <map>  #include <SFML/Graphics.hpp>  #include <iostream>  #include "tinyxml.h"  struct Object  {  int GetPropertyInt(std::string name);//номер свойства объекта в нашем списке  float GetPropertyFloat(std::string name);  std::string GetPropertyString(std::string name);  std::string name;//объявили переменную name типа string  std::string type;//а здесь переменную type типа string  sf::Rect<float> rect;//тип Rect с нецелыми значениями  std::map<std::string, std::string> properties;//создаём ассоциатиный массив. ключ - строковый тип, значение - строковый  sf::Sprite sprite;//объявили спрайт  };  struct Layer//слои  {  int opacity;//непрозрачность слоя  std::vector<sf::Sprite> tiles;//закидываем в вектор тайлы  };  class Level//главный класс - уровень  {  public:  bool LoadFromFile(std::string filename);//возвращает false если не получилось загрузить  Object GetObject(std::string name);  std::vector<Object> GetObjects(std::string name);//выдаем объект в наш уровень  std::vector<Object> GetAllObjects();//выдаем все объекты в наш уровень  void Draw(sf::RenderWindow &window);//рисуем в окно  sf::Vector2i GetTileSize();//получаем размер тайла  private:  int width, height, tileWidth, tileHeight;//в tmx файле width height в начале,затем размер тайла  int firstTileID;//получаем айди первого тайла  sf::Rect<float> drawingBounds;//размер части карты которую рисуем  sf::Texture tilesetImage;//текстура карты  std::vector<Object> objects;//массив типа Объекты, который мы создали  std::vector<Layer> layers;  };  ///////////////////////////////////////  int Object::GetPropertyInt(std::string name)//возвращаем номер свойства в нашем списке  {  return atoi(properties[name].c\_str());  }  float Object::GetPropertyFloat(std::string name)  {  return strtod(properties[name].c\_str(), NULL);  }  std::string Object::GetPropertyString(std::string name)//получить имя в виде строки.вроде понятно  {  return properties[name];  }  bool Level::LoadFromFile(std::string filename)//двоеточия-обращение к методам класса вне класса  {  TiXmlDocument levelFile(filename.c\_str());//загружаем файл в TiXmlDocument  // загружаем XML-карту  if (!levelFile.LoadFile())//если не удалось загрузить карту  {  std::cout << "Loading level \"" << filename << "\" failed." << std::endl;//выдаем ошибку  return false;  }  // работаем с контейнером map  TiXmlElement \*map;  map = levelFile.FirstChildElement("map");  // пример карты: <map version="1.0" orientation="orthogonal"  // width="10" height="10" tilewidth="34" tileheight="34">  width = atoi(map->Attribute("width"));//извлекаем из нашей карты ее свойства  height = atoi(map->Attribute("height"));//те свойства, которые задавали при работе в  tileWidth = atoi(map->Attribute("tilewidth"));//тайлмап редакторе  tileHeight = atoi(map->Attribute("tileheight"));  // Берем описание тайлсета и идентификатор первого тайла  TiXmlElement \*tilesetElement;  tilesetElement = map->FirstChildElement("tileset");  firstTileID = atoi(tilesetElement->Attribute("firstgid"));  // source - путь до картинки в контейнере image  TiXmlElement \*image;  image = tilesetElement->FirstChildElement("image");  std::string imagepath = image->Attribute("source");  // пытаемся загрузить тайлсет  sf::Image img;  if (!img.loadFromFile(imagepath))  {  std::cout << "Failed to load tile sheet." << std::endl;//если не удалось загрузить тайлсет-выводим ошибку в консоль  return false;  }  img.createMaskFromColor(sf::Color(255, 255, 255));//для маски цвета.сейчас нет маски  tilesetImage.loadFromImage(img);  tilesetImage.setSmooth(false);//сглаживание  // получаем количество столбцов и строк тайлсета  int columns = tilesetImage.getSize().x / tileWidth;  int rows = tilesetImage.getSize().y / tileHeight;  // вектор из прямоугольников изображений (TextureRect)  std::vector< sf::Rect<int> > subRects;  for (int y = 0; y < rows; y++)  for (int x = 0; x < columns; x++)  {  sf::Rect<int> rect;  rect.top = y \* tileHeight;  rect.height = tileHeight;  rect.left = x \* tileWidth;  rect.width = tileWidth;  subRects.push\_back(rect);  }  // работа со слоями  TiXmlElement \*layerElement;  layerElement = map->FirstChildElement("layer");  while (layerElement)  {  Layer layer;  // если присутствует opacity, то задаем прозрачность слоя, иначе он полностью непрозрачен  if (layerElement->Attribute("opacity") != NULL)  {  float opacity = strtod(layerElement->Attribute("opacity"), NULL);  layer.opacity = 255 \* opacity;  }  else  {  layer.opacity = 255;  }  //  контейнер <data>  TiXmlElement \*layerDataElement;  layerDataElement = layerElement->FirstChildElement("data");  if (layerDataElement == NULL)  {  std::cout << "Bad map. No layer information found." << std::endl;  }  //  контейнер <tile> - описание тайлов каждого слоя  TiXmlElement \*tileElement;  tileElement = layerDataElement->FirstChildElement("tile");  if (tileElement == NULL)  {  std::cout << "Bad map. No tile information found." << std::endl;  return false;  }  int x = 0;  int y = 0;  while (tileElement)  {  int tileGID = atoi(tileElement->Attribute("gid"));  int subRectToUse = tileGID - firstTileID;  // Устанавливаем TextureRect каждого тайла  if (subRectToUse >= 0)  {  sf::Sprite sprite;  sprite.setTexture(tilesetImage);  sprite.setTextureRect(subRects[subRectToUse]);  sprite.setPosition(x \* tileWidth, y \* tileHeight);  sprite.setColor(sf::Color(255, 255, 255, layer.opacity));  layer.tiles.push\_back(sprite);//закидываем в слой спрайты тайлов  }  tileElement = tileElement->NextSiblingElement("tile");  x++;  if (x >= width)  {  x = 0;  y++;  if (y >= height)  y = 0;  }  }  layers.push\_back(layer);  layerElement = layerElement->NextSiblingElement("layer");  }  // работа с объектами  TiXmlElement \*objectGroupElement;  // если есть слои объектов  if (map->FirstChildElement("objectgroup") != NULL)  {  objectGroupElement = map->FirstChildElement("objectgroup");  while (objectGroupElement)  {  //  контейнер <object>  TiXmlElement \*objectElement;  objectElement = objectGroupElement->FirstChildElement("object");  while (objectElement)  {  // получаем все данные - тип, имя, позиция, и тд  std::string objectType;  if (objectElement->Attribute("type") != NULL)  {  objectType = objectElement->Attribute("type");  }  std::string objectName;  if (objectElement->Attribute("name") != NULL)  {  objectName = objectElement->Attribute("name");  }  int x = atoi(objectElement->Attribute("x"));  int y = atoi(objectElement->Attribute("y"));  int width, height;  sf::Sprite sprite;  sprite.setTexture(tilesetImage);  sprite.setTextureRect(sf::Rect<int>(0, 0, 0, 0));  sprite.setPosition(x, y);  if (objectElement->Attribute("width") != NULL)  {  width = atoi(objectElement->Attribute("width"));  height = atoi(objectElement->Attribute("height"));  }  else  {  width = subRects[atoi(objectElement->Attribute("gid")) - firstTileID].width;  height = subRects[atoi(objectElement->Attribute("gid")) - firstTileID].height;  sprite.setTextureRect(subRects[atoi(objectElement->Attribute("gid")) - firstTileID]);  }  // экземпляр объекта  Object object;  object.name = objectName;  object.type = objectType;  object.sprite = sprite;  sf::Rect <float> objectRect;  objectRect.top = y;  objectRect.left = x;  objectRect.height = height;  objectRect.width = width;  object.rect = objectRect;  // "переменные" объекта  TiXmlElement \*properties;  properties = objectElement->FirstChildElement("properties");  if (properties != NULL)  {  TiXmlElement \*prop;  prop = properties->FirstChildElement("property");  if (prop != NULL)  {  while (prop)  {  std::string propertyName = prop->Attribute("name");  std::string propertyValue = prop->Attribute("value");  object.properties[propertyName] = propertyValue;  prop = prop->NextSiblingElement("property");  }  }  }  objects.push\_back(object);  objectElement = objectElement->NextSiblingElement("object");  }  objectGroupElement = objectGroupElement->NextSiblingElement("objectgroup");  }  }  else  {  std::cout << "No object layers found..." << std::endl;  }  return true;  }  Object Level::GetObject(std::string name)  {  // только первый объект с заданным именем  for (int i = 0; i < objects.size(); i++)  if (objects[i].name == name)  return objects[i];  }  std::vector<Object> Level::GetObjects(std::string name)  {  // все объекты с заданным именем  std::vector<Object> vec;  for (int i = 0; i < objects.size(); i++)  if (objects[i].name == name)  vec.push\_back(objects[i]);  return vec;  }  std::vector<Object> Level::GetAllObjects()  {  return objects;  };  sf::Vector2i Level::GetTileSize()  {  return sf::Vector2i(tileWidth, tileHeight);  }  void Level::Draw(sf::RenderWindow &window)  {  // рисуем все тайлы (объекты не рисуем!)  for (int layer = 0; layer < layers.size(); layer++)  for (int tile = 0; tile < layers[layer].tiles.size(); tile++)  window.draw(layers[layer].tiles[tile]);  }  #endif // LEVEL\_H\_INCLUDED |

Реалізація: https://github.com/SorochynskyiVladyslav/CoursesRepository/blob/master/courses/prog\_base\_3/project%201.0/level.h

Main – головний модуль, що використовує інші, і власне реалізує роботу програми.

|  |
| --- |
| main.cpp |
| #include <SFML/Graphics.hpp>  #include <SFML/Audio.hpp>  #include <list>  #include <iostream>  #include <sstream>  #include <vector>  #include <math.h>  #include <windows.h>  #include "entity.h"  #include "level.h"  #include "view.h"  using namespace sf;  void youWin() {  SoundBuffer winBuffer;  winBuffer.loadFromFile("sounds/win.wav");  Sound winSound;  winSound.setBuffer(winBuffer);  Texture winText;  winText.loadFromFile("images/win.png");  Sprite win(winText);  win.setPosition(100, 30);  winSound.play();  bool isMenu = 1;  RenderWindow window(VideoMode::getDesktopMode(), "Mario 0.5", Style::Fullscreen);  while (isMenu) {  Event event;  while (window.pollEvent(event))  {  if (event.type == Event::Closed){  window.close();  }  }  if (Keyboard::isKeyPressed(Keyboard:: Escape))  return;  window.clear(Color(167, 207, 218));  window.draw(win);  window.display();  }  }  void menu () {  Texture men1, men2, men3, menBackground, menHelp;  men1.loadFromFile("images/Menu1.png");  men2.loadFromFile("images/Menu2.png");  men3.loadFromFile("images/Menu3.png");  menHelp.loadFromFile("images/Menuhelp.png");  menBackground.loadFromFile("images/MarioMenu.png");  Sprite menu1(men1), menu2(men2), menu3(men3), about(menHelp), menuBg(menBackground);  bool isMenu = 1;  int menuNum = 0;  menu1.setPosition(100, 30);  menu2.setPosition(100, 130);  menu3.setPosition(100, 230);  menuBg.setPosition(345, 0);  RenderWindow window(VideoMode::getDesktopMode(), "Mario 0.5", Style::Fullscreen);  while (isMenu) {  Event event;  while (window.pollEvent(event))  {  if (event.type == Event::Closed){  window.close();  }  }  menu1.setColor(Color::White);  menu2.setColor(Color::White);  menu3.setColor(Color::White);  menuNum = 0;  window.clear(Color(167, 207, 218));  if (IntRect(100, 30, 300, 100).contains(Mouse::getPosition(window))) {  menu1.setColor(Color::Blue);  menuNum = 1;  }  if (IntRect(100, 131, 300, 100).contains(Mouse::getPosition(window))) {  menu2.setColor(Color::Blue);  menuNum = 2;  }  if (IntRect(100, 232, 300, 100).contains(Mouse::getPosition(window))) {  menu3.setColor(Color::Blue);  menuNum = 3;  }  if (Mouse::isButtonPressed(Mouse::Left)) {  if (menuNum == 1) isMenu = false;  if (menuNum == 2) {  window.draw(about);  window.display();  while (!Keyboard::isKeyPressed(Keyboard::Escape));  }  if (menuNum == 3) {  window.close();  exit (0);  isMenu = false;  }  }  window.draw(menuBg);  window.draw(menu1);  window.draw(menu2);  window.draw(menu3);  window.display();  }  }  bool startGame (int & startLevel, RenderWindow & window, int & lives) {  view.reset(FloatRect(0, 0, 700, 600));  Level lvl;  switch (startLevel) {  case 1:  lvl.LoadFromFile("level1.tmx");  break;  case 2:  lvl.LoadFromFile("level1.tmx");  break;  case 3:  lvl.LoadFromFile("level1.tmx");  break;  }  //lvl.LoadFromFile("level1.tmx");  SoundBuffer coinBuffer;  coinBuffer.loadFromFile("sounds/coin.wav");  Sound coinSound;  coinSound.setBuffer(coinBuffer);  SoundBuffer deathBuffer;  deathBuffer.loadFromFile("sounds/mariodie.wav");  Sound deathSound;  deathSound.setBuffer(deathBuffer);  SoundBuffer stageBuffer;  stageBuffer.loadFromFile("sounds/stage.wav");  Sound stageSound;  stageSound.setBuffer(stageBuffer);  Image heroImage;  heroImage.loadFromFile("images/mario.png");  heroImage.createMaskFromColor(Color(255, 255, 255));  Image easyEnemyImage;  easyEnemyImage.loadFromFile("images/EasyEnemy.png");  easyEnemyImage.createMaskFromColor(Color(255, 255, 255));  Image coinImage;  coinImage.loadFromFile("images/coin.png");  coinImage.createMaskFromColor(Color(255, 255, 255));  Image hardEnemyImage;  hardEnemyImage.loadFromFile("images/HardEnemy.png");  hardEnemyImage.createMaskFromColor(Color(255, 255, 255));  Image mediumEnemyImage;  mediumEnemyImage.loadFromFile("images/MediumEnemy.png");  mediumEnemyImage.createMaskFromColor(Color(255, 255, 255));  Image movePlatformImage;  movePlatformImage.loadFromFile("images/MovingPlatform.png");  Font font;  font.loadFromFile("MISTRAL.TTF");  Text Score("score", font, 30), Lives("lives", font, 30);  Score.setFont(font);  Lives.setFont(font);  Score.setPosition(50, 50);  Lives.setPosition(50, 100);  Score.setColor(Color::Red);  Lives.setColor(Color::Red);  std::list<Entity\*> entities;  std::list<Entity\*>::iterator it;  std::list<Entity\*>::iterator it2;  std::vector<Object> e = lvl.GetObjects("EasyEnemy");  for (int i = 0; i < e.size(); i++) {  entities.push\_back(new Enemy(easyEnemyImage, "EasyEnemy", lvl, e[i].rect.left, e[i].rect.top, 15, 25));  }  e = lvl.GetObjects("coin");  for (int i = 0; i < e.size(); i++) {  entities.push\_back(new Coin(coinImage, "coin", lvl, e[i].rect.left, e[i].rect.top, 15, 15));  }  e = lvl.GetObjects("MediumEnemy");  for (int i = 0; i < e.size(); i++) {  entities.push\_back(new Enemy(mediumEnemyImage, "MediumEnemy", lvl, e[i].rect.left, e[i].rect.top, 16, 24));  }  e = lvl.GetObjects("HardEnemy");  for (int i = 0; i < e.size(); i++) {  entities.push\_back(new Enemy(hardEnemyImage, "HardEnemy", lvl, e[i].rect.left, e[i].rect.top, 24, 24));  }  e = lvl.GetObjects("exit");  for (int i = 0; i < e.size(); i++) {  entities.push\_back(new Exit(mediumEnemyImage, "exit", lvl, e[i].rect.left, e[i].rect.top, 20, 25));  }  e = lvl.GetObjects("fire");  for (int i = 0; i < e.size(); i++) {  entities.push\_back(new Fire(mediumEnemyImage, "fire", lvl, e[i].rect.left, e[i].rect.top, 200, 25));  }  e = lvl.GetObjects("MovingPlatform");  for (int i = 0; i < e.size(); i++)  entities.push\_back(new MovingPlatform(movePlatformImage, "MovingPlatform", lvl, e[i].rect.left, e[i].rect.top, 95, 22));  Object player = lvl.GetObject("player");  Player p(heroImage, "Player", lvl, player.rect.left, player.rect.top, 22, 38);  Clock clock;  while (window.isOpen())  {  float time = clock.getElapsedTime().asMicroseconds();  clock.restart();  time = time / 800;  Event event;  while (window.pollEvent(event))  {  if (event.type == sf::Event::Closed)  window.close();  }  for (it = entities.begin(); it != entities.end();) {  Entity \*b = \*it;  b->update(time);  if (b->life == false) {  it = entities.erase(it);  delete b;  }  else it++;  }  for (it = entities.begin(); it != entities.end(); it++) {  if (((\*it)->name == "MovingPlatform") && ((\*it)->getRect().intersects(p.getRect())))  {  Entity \*movPlat = \*it;  if (p.dy>0)  if (p.y + p.height < movPlat->y + movPlat->height)  {  p.y = movPlat->y - p.height + 3;  p.x += movPlat->dx\*time;  p.dy = 0;  p.onGround = true;  }  }  if ((\*it)->getRect().intersects(p.getRect())) {  if ((\*it)->name == "EasyEnemy" || (\*it)->name == "MediumEnemy" || (\*it)->name == "HardEnemy") {  if ((p.dy>0) && fabs(p.x - (\*it)->x) <= 25 && (p.onGround == false) && (p.y - p.height < (\*it)->y - (\*it)->height)) {  p.dy = -0.3;  (\*it)->lives--;  if ((\*it)->name == "EasyEnemy")  (\*it)->sprite.setTextureRect(IntRect(60, 8, 16, 8));  p.Score++;  }  else {  if (((\*it)->lives != 0)) {  deathSound.play();  p.Score = 0;  p.lives--;  lives--;  Sleep(2000);  return false;  }  }  }  if ((\*it)->name == "fire") {  deathSound.play();  p.Score = 0;  p.lives--;  lives--;  Sleep(2000);  return false;  }  if ((\*it)->name == "coin") {  p.Score++;  coinSound.play();  (\*it)->life = false;  }  if ((\*it)->name == "exit") {  stageSound.play();  startLevel++;  Sleep(3000);  return false;  }  }  for (it2 = entities.begin(); it2 != entities.end(); it2++) {  if ((\*it)->getRect() != (\*it2)->getRect())  if (((\*it)->getRect().intersects((\*it2)->getRect())) && ((\*it)->name == "EasyEnemy")){  (\*it)->dx \*= -1;  (\*it)->sprite.scale(-1, 1);  }  }  }  ostringstream playerScoreString;  playerScoreString << p.Score;  ostringstream playerLivesString;  playerLivesString << lives;  p.update(time);  if (Keyboard::isKeyPressed(Keyboard::Escape)) {  lives = 0;  return false;  }  window.setView(view);  window.clear(Color(167, 207, 218));  lvl.Draw(window);  for (it = entities.begin(); it != entities.end(); it++) {  window.draw((\*it)->sprite);  }  Score.setString("Score: " + playerScoreString.str());  Lives.setString("Lives: " + playerLivesString.str());  Score.setPosition(view.getCenter().x - 350, view.getCenter().y - 250);  Lives.setPosition(view.getCenter().x - 350, view.getCenter().y - 300);  window.draw(Score);  window.draw(Lives);  window.draw(p.sprite);  window.display();  }  return 0;  }  void gameRunning (int & startLevel, RenderWindow & window, int & lives) {  if (startGame (startLevel, window, lives)) gameRunning(startLevel, window, lives);  else {  if (startLevel == 4) {  youWin();  return;  }  if (lives == 0) {  lives = 3;  startLevel = 1;  menu();  return;  }  if (lives != 0) {  gameRunning (startLevel, window, lives);  }  }  }  int main()  {  RenderWindow window(VideoMode::getDesktopMode(), "Mario 0.5", Style::Fullscreen);  int StartLevel = 1;  int lives = 3;  menu();  while (1) {  StartLevel = 1;  lives = 3;  gameRunning(StartLevel, window, lives);  menu();  }  return 0;  } |

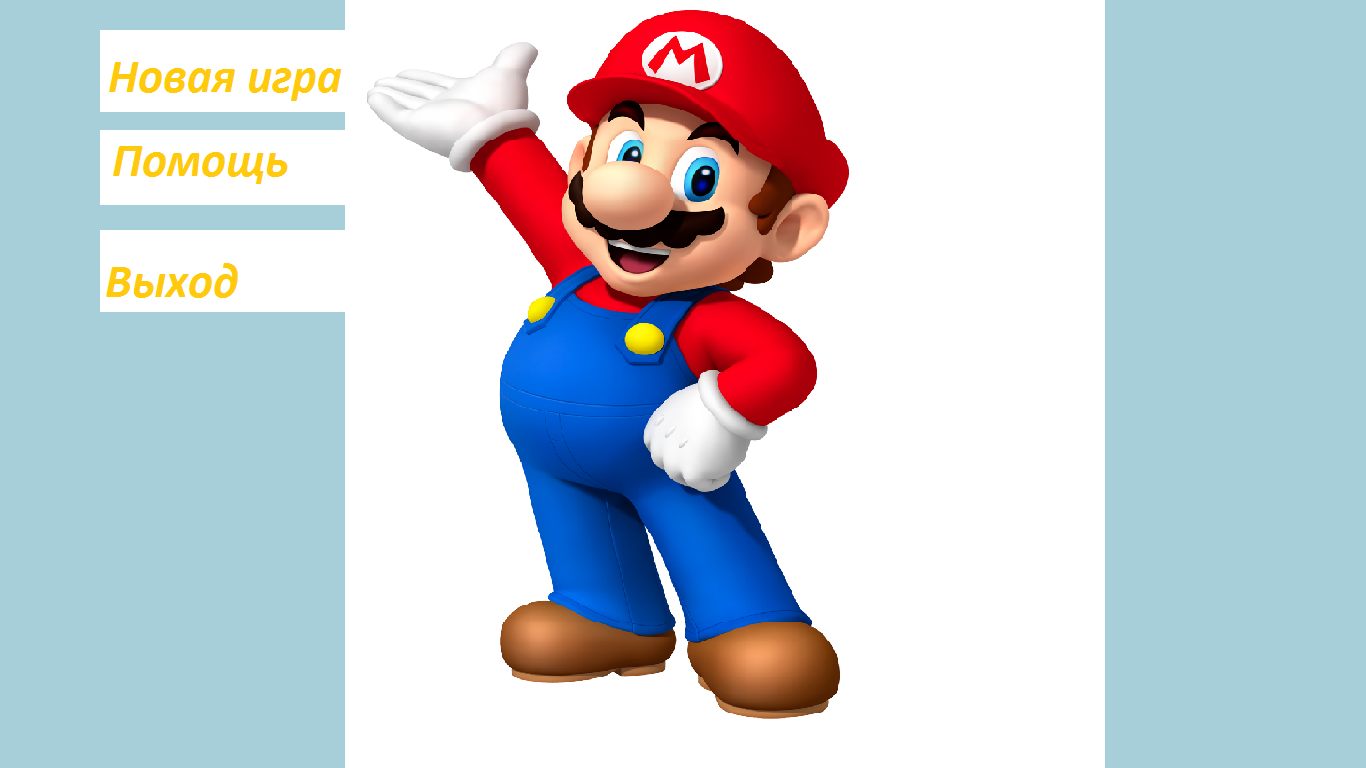
Реалізація: https://github.com/SorochynskyiVladyslav/CoursesRepository/blob/master/courses/prog\_base\_3/project%201.0/view.h

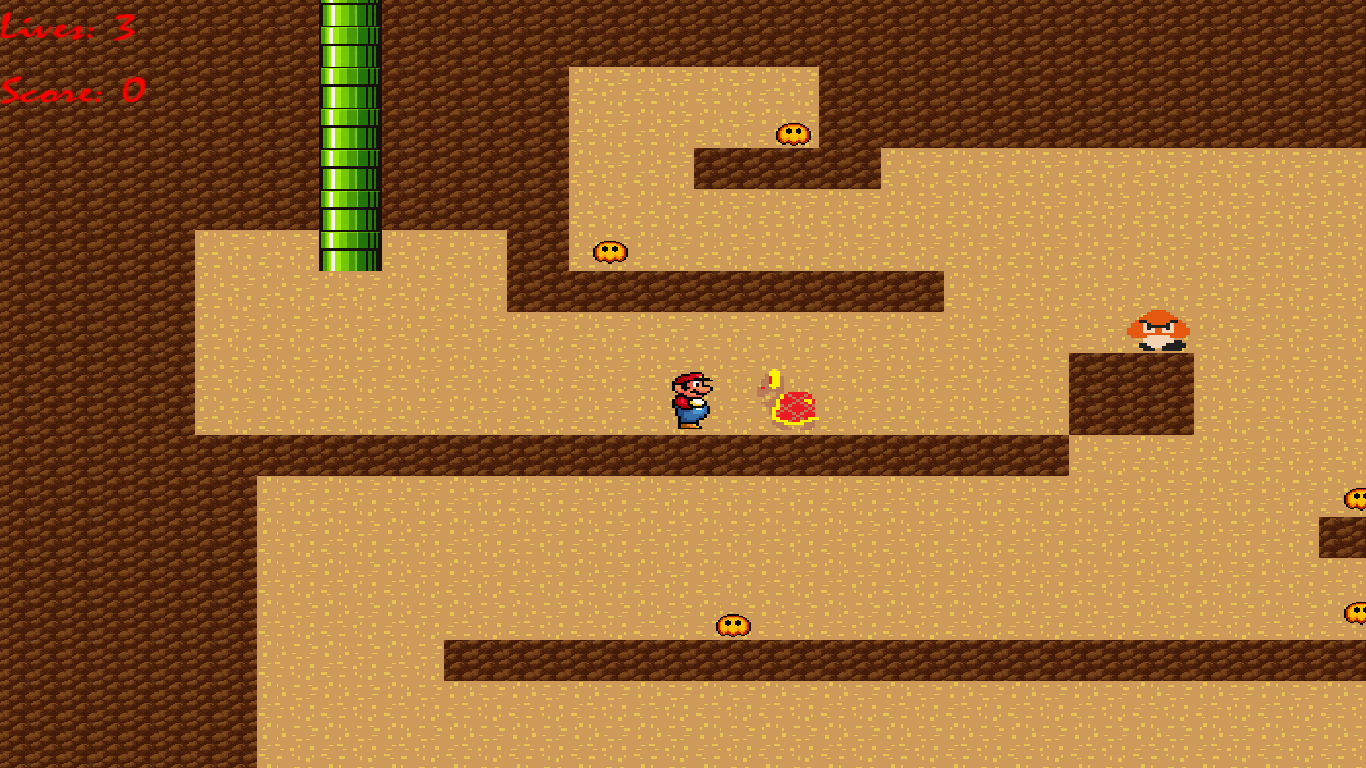
**Опис форматів файлів**

Tiled map File (TMX) — формат зберігання карти.

Файл обміну даних Translation Memory, що використовує стандарт xml.

**Результати роботи**





**Короткий мануал для користувача**

Управління відбувається за допомогою клавіатури:

1. Стрілка вправо – йти вперед.

2. Стрілка вліво – йти назад.

3. Стрілка вгору – стрибнути.