CPROG Rapport för Programmeringsprojektet

[Gruppnummer: 26]
[Gruppmedlemmar: Khaled Alraas 20011510 - 2873]

Skriv en kortfattad instruktion för hur programmeringsprojektet skall byggas och testas, vilka krav som måste vara uppfyllda, sökvägar till resursfiler(bildfiler/ljudfiler/typsnitt mm), samt vad en spelare förväntas göra i spelet, hur figurernas rörelser kontrolleras, mm.
Om avsteg gjorts från kraven på Filstruktur, så måste också detta motiveras och beskrivas i rapporten.

Fyll i 'check-listan', så att du visar att du tagit hänsyn till respektive krav, skriv också en kort kommentar om på vilket sätt du/gruppen anser att kravet tillgodosetts, och/eller var i koden kravet uppfylls.

Den ifyllda Rapportmallen lämnas in tillsammans med Programmeringsprojektet. Spara rapporten som en PDF med namnet CPROG_RAPPORT_GRUPP_NR.pdf (där NR är gruppnumret).

1. Beskrivning

orsaker vad som ska skrivas här men..
Mitt spelmotor bygger på …
Component som håller koll på Position och Size och behandlar SDL_Rect
TextureManager håller koll texture att dess basklasser implementerar createTexture().
TextManager ärver från **TextureManager** och tar ansvar att skapa text texture med(font, fontSize, text, color).
ImageManager ärver från **TextureManager** och tar ansvar att skapa bild texture.
Label ärver från **TextManager** och **Component,** dess roll är att visa upp text.
Image ärver från **ImagetManager** och **Component,** dess roll är att visa upp bilder.
Conatiner som ärver från **Component,** dess ansvar är att hålla components i en std::vector och dessa components ska flyttas med container.
BoxCollider som ärver från **Conatiner ,** med hjälp ut av System klass denna klass lyssnar på kollision och låter användaren göra något åt det.
Session är ansvarig att hålla spelet genom att hantera i sin while loop Events som till exempel inputs, sen fysiken och detta gäller update funktion som användaren får bestämma och även Collision, efter det så renderas resultatet på skärmen. Session också ansvarig för vad som ska visas och vad för FPS ska detta visas på.

2. Instruktion för att bygga och testa

NOTE: **TextureManager.h** in my GameEngine is the same as **Constants.h** from your description.

Game Engine Basic Instructions

- Create an Image/Sprite

Image::getInstance takes the name of the image you want to show on the screen.

```
int main(int argc, char** argv) {
    Session ses;
    Image* player = Image::getInstance("player.png");
    ses.add(player);
    ses.run();
```

- Create an Image/Sprite with BoxCollider

BoxCollider will automatically resize it self so that the element/s fits inside it

```
int main(int argc, char** argv) {
    Session ses;
    Image* player = Image::getInstance("player.png");
    BoxCollider* boxCollider = BoxCollider::getInstance();
    boxCollider->add(player);
    ses.add(player);
    ses.run();
```

- Make Image/Sprite move

Using **ses.update** to update variables values such as the players position.
NOTE: make sure you move the **BoxCollider** and not the actual Image. The **BoxCollider** will automatically move the Components/items inside it with it.
NOTE: you do not need a **BoxCollider** in order to move your **Image**.

```
int main(int argc, char** argv) {
    Session ses;
    Image* player = Image::getInstance("player.png");
    BoxCollider* boxCollider = BoxCollider::getInstance();
    boxCollider->add(player);
    ses.add(player);
    ses.update([&boxCollider/*put the address of the vairable you want to reference*/] {
        Position posTemp(boxCollider->getPosition().x + 1, boxCollider->getPosition().y);
        boxCollider->setPosition(posTemp);
        });
    ses.run();
    return 0;
}
```

- Collision

In order for collision to occur you need two objects with **BoxCollider** to collide with each other. When that happens you can simply **ignore it** or to write a **onCollisionEnter** function where you decide what to do. You get to know who did you collide with using "**other**".

```
boxCollider->onCollisionEnter([&canMove](BoxCollider* other) {
    if (other->getName() == "wall") canMove = false;
    });
```

- Input

Using **Session** you have the choice of doing something when the user enters a specific key. In the case of **KeyBoard** you need to specify which **keyboard key** you want to use. In the case of the **Mouse** you need to specify which **mouse button** you want to use + you get to know the **position** of where the click happens.

```
ses.addKeyDownCallback(SDLK_RIGHT, [&moveRight, &canMoveLeft] {
    moveRight = true;
    canMoveLeft = true;
    });
ses.addKeyUpCallback(SDLK_RIGHT, [&moveRight] {
    moveRight = false;
    });
ses.addKeyDownCallback(SDLK_LEFT, [&moveLeft, &canMoveRight] {
    moveLeft = true;
    canMoveRight = true;
    });
ses.addKeyUpCallback(SDLK_LEFT, [&moveLeft] {
    moveLeft = false;
    });
ses.addMouseDownButtonCallback(SDL_BUTTON_LEFT, [](int x, int y) {
    std::cout << "SHOOT!";
    });
```

- Change Directory for images and fonts by going to TextureManager and change them there.

```
class TextureManager {
public:
    virtual ~TextureManager();
    const SDL_Texture* getTexture();
protected:
    virtual void createTexture() = 0;
    SDL_Texture* texture = nullptr;
    std::string imagesAddress = "C:/Development/GameEngine 2/resources/images/";
    std::string fontsAddress = "C:/Development/GameEngine 2/resources/fonts/";
};
```

Game Instructions

NOTE: there is only one script for the actual game it is **MyGame.CPP**

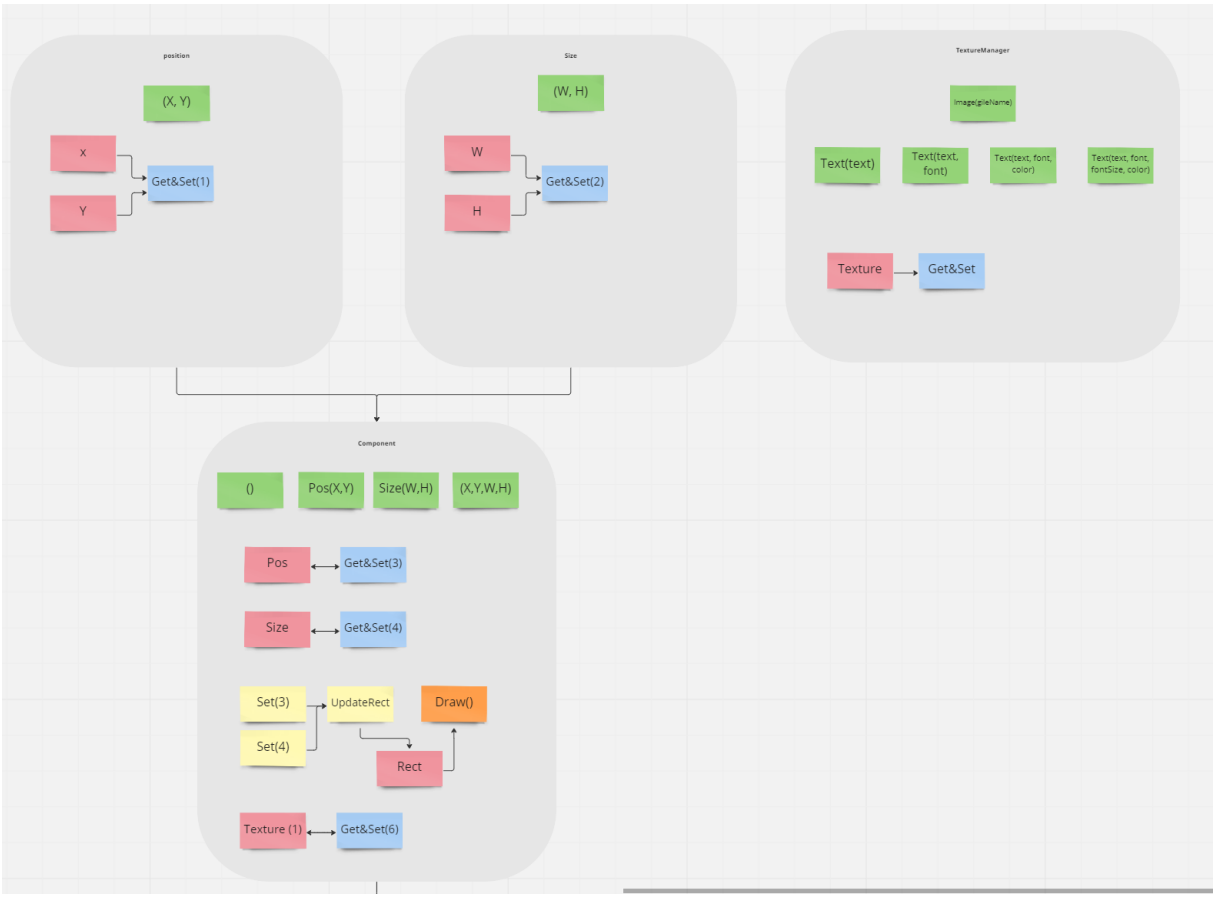
- The player can go right or left using the **arrow keys** right and left.
- The player can shoot bullets with the **spacebar** keyboard key.
- The enemy spawns at a random position from the top of the screen. The enemy keeps falling down.

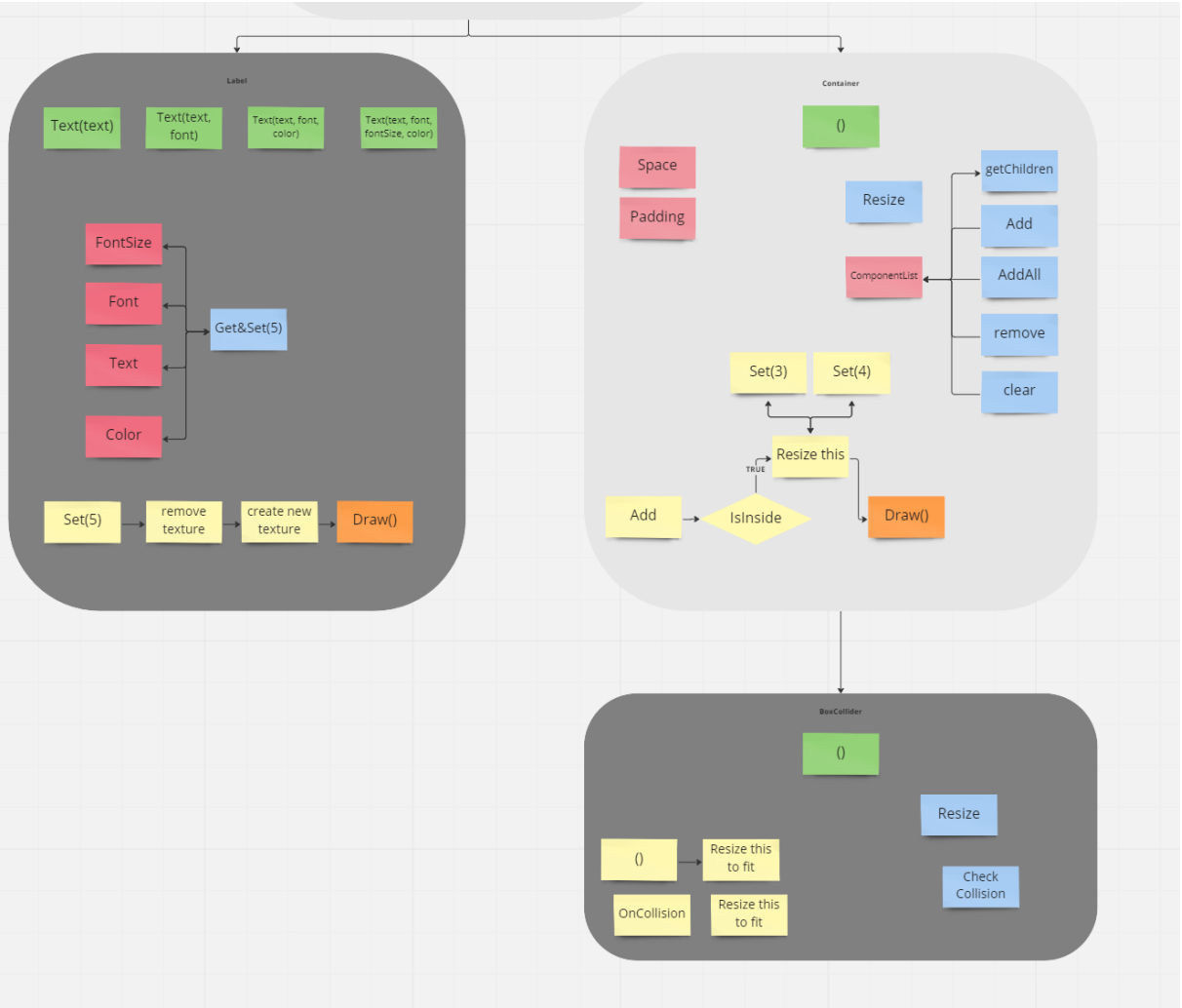
- If the enemy touches you, you lose and the application closes.

3. Krav på den Generella Delen(Spelmotorn)

3.1. [Ja/Nej/Delvis] Programmet kodas i C++ och grafikbiblioteket SDL2 används.
Kommentar: Ja, allt är byggt med hjälp ut av SDL2 grafikbiblioteket (SDL, SDL_image, SDL_ttf)

3.2. [Ja/Nej/Delvis] Objektorienterad programmering används, dvs. programmet är uppdelat i klasser och använder av oo-tekniker som inkapsling, arv och polymorfism.
Kommentar: Ja absolut jag även gjort en program code arkitektur. Kolla på bilderna nedan och **notera** att det är en lite gammal version jämfört med hur ser det ut nu + den följer form av utseende jag kom på.


position
(X, Y)
x
Y
Get&Set(1)
Size
(W, H)
W
H
Get&Set(2)
TextureManager
Image(fileName)
Text(text)
Text(text, font)
Text(text, font, color)
Text(text, font, fontSize, color)
Texture
Get&Set
Component
()
Pos(X,Y)
Size(W,H)
(X,Y,W,H)
Pos
Get&Set(3)
Size
Get&Set(4)
Set(3)
Set(4)
UpdateRect
Draw()
Rect
Texture (1)
Get&Set(6)



3.3. [Ja/Nej/Delvis] Tillämpningsprogrammeraren skyddas mot att använda värdesemantik för objekt av polymorfa klasser.
Kommentar: Ja, Man skapar komponenterna på heapen och använder värdesemantik för att ta och skicka data.

3.4. [Ja/Nej/Delvis] Det finns en gemensam basklass för alla figurer(rörliga objekt), och denna basklass är förberedd för att vara en rotklass i en klasshierarki.
Kommentar: Ja, jag har redan visat det i bilden ovan. Mina root klasser är **Component** och **Texturemanager**. t.ex. **Image** klass ärver från **Component** och **TextManager** som i sin tur ärver från **TextureManager .**

3.5. [Ja/Nej/Delvis] Inkapsling: datamedlemmar är privata, om inte ange skäl.
Kommentar: Förutom Position och Size klasser, alla Klasser har variablerna antingen **privata** eller **protected** och för de har jag använder public **Getters** och **Setters**.

3.6. [Ja/Nej/Delvis] Det finns inte något minnesläckage, dvs. jag har testat och sett till att dynamiskt allokerat minne städas bort.

Kommentar: Jag försökte mitt bästa att behandla minnesläckage. Men jag känner att jag fixade de flesta. Jag tror **alla klasser hanterar sin pointers korrekt**, men det är **kanske möjligt** att **själva spelet** innehåller minnesläckage.

3.7. [Ja/Nej/Delvis] Spelmotorn kan ta emot input (tangentbordshändelser, mushändelser) och reagera på dem enligt tillämpningsprogrammets önskemål, eller vidarebefordra dem till tillämpningens objekt.
Kommentar: Ja absolut. Man kan använda både mus och tangentbord.

3.8. [Ja/Nej/Delvis] Spelmotorn har stöd för kollisionsdetektering: dvs. det går att kolla om en Sprite har kolliderat med en annan Sprite.
Kommentar: Ja absolut. Det finns en kollidera som man kan använda på det sättet man vill.

3.9. [Ja/Nej/Delvis] Programmet är kompilerbart och körbart på en dator under både Mac, Linux och MS Windows (alltså inga plattformspecifika konstruktioner) med SDL 2 och SDL2_ttf, SDL2_image och SDL2_mixer.
Kommentar: Jag antar att det funkar inga problem. Jag undvider användning av **pragma once** och istället så använder jag **ifndef … define … endif**, kan inte riktigt testa det.

4. Krav på den Specifika Delen(Spelet som använder sig av Spelmotorn)

4.1. [Ja/Nej/Delvis] Spelet simulerar en värld som innehåller olika typer av visuella objekt. Objekten har olika beteenden och rör sig i världen och agerar på olika sätt när de möter andra objekt.
Kommentar: Ja, jag kolla på de kommentarerna för frågorna nedåt.

4.2. [Ja/Nej/Delvis] Det finns minst två olika typer av objekt, och det finns flera instanser av minst ett av dessa objekt.
Kommentar: Det finns Wall, Bullet, Player, och Enemy

4.3. [Ja/Nej/Delvis] Figurerna kan röra sig över skärmen.
Kommentar: Ja man kan styra spelaren och fienderna rör nedåt och Bullets rör uppåt.

4.4. [Ja/Nej/Delvis] Världen (spelplanen) är tillräckligt stor för att den som spelar skall uppleva att figurerna förflyttar sig i världen.
Kommentar: spelet är byggt för en fixed skärm så att inget som är relevant går utanför skärmen.

4.5. [Ja/Nej/Delvis] En spelare kan styra en figur, med tangentbordet eller med musen.
Kommentar: spelaren kan gå höger och vänster och skjuter med SpaceBar

4.6. [Ja/Nej/Delvis] Det händer olika saker när objekten möter varandra, de påverkar varandra på något sätt.
Kommentar: spelaren kolliderar med vägarna och kan inte gå genom de. spelaren dör om en fiende “touches him”. fienden dör när bullet “touches him”.