CPROG Rapport för Programmeringsprojektet
[Gruppnummer: 05 ]
[Gruppmedlemmar: Jacob Hjorth - 19961029-0372 ]
*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).

## Beskrivning
Jag gjorde två olika spel för att kunna testa att spelmotorn fungerade för olika typer av spel. Spelet Starship är det spelet som jag är mest nöjd med, och som du som testare borde köra först.

### Starship
I Starship ska du som spelare skjuta ner, alternativt fånga fiender med din “Pistol”. Fienderna kommer flygandes uppifrån och som rör sig mot dig i y-led. Ditt mål är att skjuta ner/fånga 20 st fiender. Om 3 st fiender lyckas ta sig förbi dig så förlorar du. Du kontrollerar din pistol med musen, och skjuter genom att klicka på musen.

### RabbitHammer
I RabbitHammer ska du som spelare hamra med “Sledgehammer” på fiender som dyker upp på skärmen. Fienderna kan dyka upp på olika platser på skärmen. Målet är att hamra 20 st fiender. Om du inte hamrar fienderna under en viss tid så kommer de försvinna. Om 3 st fiender försvinner innan du hamrat dem så kommer du förlora. Du kontrollerar din sledgehammer med musen, och hamrar genom att klicka på musen.

## Instruktion för att bygga och testa

### Sökvägar
Resursfilerna ligger i resources-mappen i projektet och alla sprites använder sig av en “constants.h”-fil för att hitta mappen. I resources-mappen så finns det tre mappar(fonts, images, sounds).

Ex (för att hitta filer i images-mappen):
```
shipTexture = IMG_LoadTexture(sys.ren, (constants::gResPath +
"images/Enemy.png").c_str());
```

### Starship
I **första steget** för att bygga och testa spelet **Starship** ska du i **Main,** initiera en **Pistol**, en **EnemySpawner** och en **HealthHandler** genom **getInstance()**. Och sedan lägga till dessa Sprites genom funktionen **addSprite()** som ligger i **GameEngine**. Du ska sedan kalla på **run()**, som också ligger i **GameEngine**.

**Andra steget** är att du går in i **EnemySpawner** och lägger till **randomSpawnShip()** i funktionen **tick().**

**Sista steget** är att gå in i **HealthHandler** och sätta **currentHealth, totalHealth, enemiesToKillTxt** och **enemiesToKill**. Detta sätter hälsan på spelaren, hur många fiender man behöver döda samt en sträng för fiendens liv. Strängen för spelarens liv behöver man inte ändra på eftersom det alltid ska stå “Health”.

**Main**
```
int main(int argc, char **argv)
{
   Pistol *pistol = Pistol::getInstance();
   EnemySpawner* spawner = EnemySpawner::getInstance();
   HealthHandler *handler = HealthHandler::getInstance();
```

```
   ge.addSprite(pistol);
   ge.addSprite(spawner);
   ge.addSprite(handler);
   ge.run();

   return 0;
}
```

**EnemySpawner**

```
void EnemySpawner::EnemySpawner::tick()
{
   randomSpawnShip();
}
```

**HealthHandler**

```
HealthHandler::HealthHandler() : Sprite(100, 100, 100, 100){
   // Set total health
   ge.setTotalHealth(3);

   // Set current health
   ge.setCurrentHealth(3);

   // Set enemies to kill string
   ge.setEnemiesToKillTxt("Ships to destroy: ");

   // Set enemies to kill
   ge.setEnemiesToKill(20);
}
```

## RabbitHammer

I **första steget** för att bygga och testa spelet **RabbitHammer** ska du i **Main,** initiera en **Sledgehammer** och en **EnemySpawner** genom **getInstance()**. Och sedan lägga till dessa Sprites genom funktionen **addSprite()** som ligger i **GameEngine**. Du ska sedan kalla på **run()**, som också ligger i **GameEngine**.

**Andra steget** är att du går in i **EnemySpawner** och lägger till **randomSpawnRabbit()** i funktionen **tick().**

**Sista steget** är att gå in i **HealthHandler** och sätta **currentHealth, totalHealth, enemiesToKillTxt** och **enemiesToKill**. Detta sätter hälsan på spelaren, hur många fiender man behöver döda samt en sträng för fiendens liv. Strängen för spelarens liv behöver man inte ändra då eftersom det alltid ska stå “Health”.

**Main**

```
int main(int argc, char **argv)
{
   Sledgehammer* sledgehammer = Sledgehammer::getInstance();
   EnemySpawner* spawner = EnemySpawner::getInstance();
   HealthHandler *handler = HealthHandler::getInstance();
   ge.addSprite(sledgehammer);
   ge.addSprite(spawner);
   ge.addSprite(handler);
   ge.run();

   return 0;
}
```

**EnemySpawner**

```
void EnemySpawner::EnemySpawner::tick()
```

```
{
   randomSpawnRabbit();
}
```

**HealthHandler**

```
HealthHandler::HealthHandler() : Sprite(100, 100, 100, 100){
   // Set total health
   ge.setTotalHealth(3);

   // Set current health
   ge.setCurrentHealth(3);

   // Set enemies to kill string
   ge.setEnemiesToKillTxt("Rabbits to hit: ");

   // Set enemies to kill
   ge.setEnemiesToKill(20);
}
```

- Krav på den Generella Delen(Spelmotorn)
    - [ Ja/Nej/Delvis ] Programmet kodas i C++ och grafikbiblioteket SDL2 används.
      Kommentar: Ja, jag har skrivit programmet i VSCode i C++ och använt mig av SDL2.

    - [ Ja/Nej/Delvis ] Objektorienterad programmering används, dvs. programmet är uppdelat i klasser och använder av oo-tekniker som inkapsling, arv och polymorfism.
      Kommentar: Ja, alla subklasser utgår från basklassen Sprite. Där jag följer tekniker för inkaplsing, arv och polymorfism.

    - [ Ja/Nej/Delvis ] Tillämpningsprogrammeraren skyddas mot att använda värdesemantik för objekt av polymorfa klasser.
      Kommentar: Ja, jag gör det genom att exempelvis sätta copy-konstruktorn och tilldelningsoperatorn private för Sprite.

    - [ 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 använder mig av “Sprite” som basklass/rotklass.

    - [ Ja/Nej/Delvis ] Inkapsling: datamedlemmar är privata, om inte ange skäl.
      Kommentar: Delvis, alla datamedlemmar är privata förutom SCREEN_HEIGHT och SCREEN_WIDTH i System.h. Detta är för att kunna hålla koll på rutans höjd och bredd. Men de är `static const` så man kan inte ändra dem. Jag har också SDL_Rect rect som protected i Sprite, för att kunna göra direkta ändringar på positionen för subklasser till Sprite.

    - [ Ja/Nej/Delvis ] Det finns inte något minnesläckage, dvs. jag har testat och sett till att dynamiskt allokerat minne städas bort.
      Kommentar: Ja, jag använder mig av vektorer added och removed för att kontrollera detta. Där jag lägger till borttagna objekt i removed, genom removeSprite(). Jag tar sedan bort objekten i removed genom en loop.

    - [ 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, jag använder mig av oliksa cases för att hantera input från användaren.

- [ Ja/Nej/Delvis ] Spelmotorn har stöd för kollisionsdetektering: dvs. det går att kolla om en Sprite har kolliderat med en annan Sprite.
  Kommentar: Ja, jag använder mig av checkCollision() för detta.

- [ 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: Ja, jag har inte använt några plattformsspecifika konstruktioner.

- Krav på den Specifika Delen(Spelet som använder sig av Spelmotorn)
  - [ 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, båda spelen har detta implementerat.

  - [ Ja/Nej/Delvis ] Det finns minst två olika typer av objekt, och det finns flera instanser av minst ett av dessa objekt.
    Kommentar: Ja, båda spelen har detta implementerat.

  - [ Ja/Nej/Delvis ] Figurerna kan röra sig över skärmen.
    Kommentar: Ja, båda spelen har detta implementerat.

  - [ Ja/Nej/Delvis ] Världen (spelplanen) är tillräckligt stor för att den som spelar skall uppleva att figurerna förflyttar sig i världen.
    Kommentar: Ja, båda spelen har detta implementerat.

  - [ Ja/Nej/Delvis ] En spelare kan styra en figur, med tangentbordet eller med musen.
    Kommentar: Ja, båda spelen kan styras med musen. Jag har också gjort en restart-skärm som styrs av tankentbordet, specifikt ENTER, alternativt att man kryssar rutan med musen.

  - [ Ja/Nej/Delvis ] Det händer olika saker när objekten möter varandra, de påverkar varandra på något sätt.
    Kommentar: Ja, det tas bort i båda spelen.