TDDC76 - Programmering och Datastrukturer

# Objektorienterad Analys

**Aqua Invaders**

**Version 2.0**


Anmar Karmush, anmka989@student.liu.se

Nils-Ruben Skogemyr, nilsk758@student.liu.se

Sigge Rystedt, sigry751@student.liu.se

Isak Wernroth, isawe536@student.liu.se

Clara Ekbäck claek609@student.liu.se

Emil Eriksson emier604@student.liu.se




Institutionen för datavetenskap (IDA)
Programvara och System (SAS)
Linköpings Universitet


December 9, 2023

# Contents

# 1. Ändringar

Här skrivs ändring som har gjorts mellan versionerna.

**Version 2:**
Figur 2.1 har uppdaterats för att ta hänsyn till de ändringar vi har gjorts. Kapitel 3 har uppdaterats så klasserna Power-up och Power-down har slagits ihop till en klass och klasserna Button, Font och GameDataReader har lagts till. Kapitel 4 har uppdaterats med ett mer specifikt användnigsfall som täcker aspekter från hela spelet. Alltså från meny tills att spelaren dör.

# 2. Klassdiagram

Figur 2.1 visar en preliminär skiss på ett UML diagram för spelet. Klassdiagrammet har uppdateras en hel del sen version 1.0 men är fortfarande inte helt bestämt än. Exempelvis kan det uppkomma mer variabler i Context klassen. Även Font klassen har vi inte helt bestämt än hur den ska implementeras. Notera att State Machine och Objekt klasserna är abstrakta klasser. Alltså kommer instanser av dessa klasser inte kunna skapas.

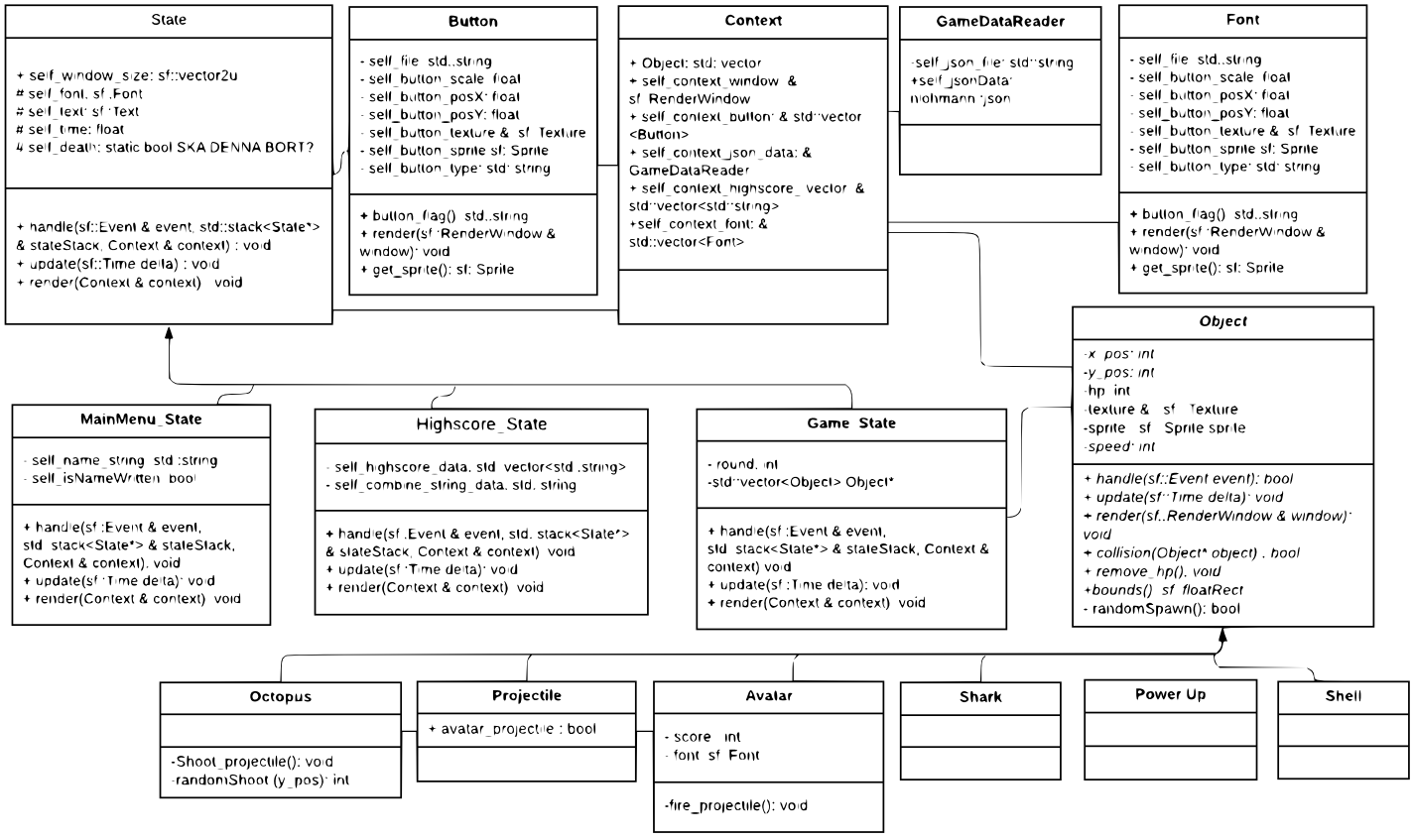


Figure 2.1: Preliminär skiss på UML diagramet för spelet.

# 3. Klassbeskrivningar

I det här kapitlet beskrivs alla klasser som finns i spelet.

## 3.1 State

State klassen kommer vara en abstrakt klass som innehåller metoder för att hantera mus och tangentbord input, uppdatera all logik i spelet samt rita ut all objekt på skärmen. Olika states kommer sparas i en State stack för att göra det enkelt att byta mellan olika states.

## 3.2 Context

Context innehåller variabler som flera klasser behöver känna till för att undvika globala variabler. Det underlättar skapande och borttagning av variabler utan minnesläckor. Det blir också enklare att lägga till variabler som flera klasser använder eftersom variablerna bara behöver läggas till i Context istället för att skicka variabler som referensparameter i varenda metod.

## 3.3 Button

Button är en klass för de olika knappar man ska kunna trycka på. Klassen innehåller det som behövs för att kunna rita ut en knapp på skärmen och avgöra om användaren tryckt på knappen. Alla kanppar har en flagga för att veta om de tryckts på eller inte.

## 3.4 Font

Font är en klass för de olika texter som kommer skrivas ut till spelaren. . Exempelvis innehåller klassen position, en skala och en textsträng som ska skrivas ut.

## 3.5 GameDataReader

GameDataReader är en klass som hanterar använingen av json-filer.

## 3.6 Game State

Game state klasen används framförallt för att starta igång spelet och även hålla kolla på vad för objekt ska finnas i spelet. Detta görs genom en vektor som innehåller lagrar alla objekt som behövs för spelet. Game State ärver från State Machine klassen.

## 3.7 MainMenu State

Main menu är en klass med en meny för att spara namn, starta spelet och visa high score. MainMenuState ärver från State Machine. State Machine stacken kommer alltid börja med en MainMenuState och beroende på vilken knapp spelaren trycker på så kan spelaren ta sig till andra states.

## 3.8 HighScore State

HighScore state kommer vara en klass med syftet att spara de tre högsta resultaten och visa en lista med dessa. HighScore State ärver från State Machine.

## 3.9 Objects

Klassen Objects kommer vara en abstrakt klass som fungerar som en basklass för spelets alla objekt. Det innefattar allt som rör sig på skärmen samt snäckan. Alla objekt kommer ha en x-kooridnat och en y-koordinat, texture och sprite för att kunna ritas ut på skärmen. Alla objekt kommer dessutom ha HP och ett objekt ska försvinna om HP = 0. Dessa värden kommer lagras på något sorts filformat så att inget är hårdkodat.

Objects kommer ha funktionen handle() för att hantera indata från användaren, update() för att updatera hastighet/texture/position och render() som ritar ut objekten på skärmen. Funktionen collision() kommer med hjälp av funktionen bounds() att kontrollera om objektet har kolliderat med något annat objekt/projektil. Slutligen finns funktionen remove_HP() som tar bort ett HP från objektet vid vissa kollisioner. Funktionen randomSpawn() skapar hajar, power-ups och power-downs med en slumpad sannolikhet. Övriga objekt kommer inte skapas med hjälp av randomSpawn() så de returnerar false. På samma sätt kommer bara avataren reagera på input och övriga objekta handle()-funktion returnerar alltid false. Notera alltså i figur 2.1 att *handle()* i Object returnerar en bool typ medans *handle()* i State Machine returnerar inget (void).

## 3.10 Shell

Shell kommer vara en klass för den statiska snäckan i spelet, som ärver egenskaper från Object. Shell ärver funktionaliteten från object, men initieras med x_speed = 0 och y_speed = 0.

### 3.11 Octopus

Octopus ärver egenskaper från Object, och känner till klassen Projectile. Klassen beskriver funktionerna för hur bläckfiskarna skjuter projektiler samt för när de ska skjuta, vilket kommer variera med avseende på dess y_pos, som är dess nuvarande y-koordinat.

### 3.12 Shark

Shark ärver funktionalitet från Obejct. Det kommer att vara en viss sannolikhet att en haj skapas längst upp på skärmen. Sannolikheten för att den skapas kommer bero på rundan och när senast en haj skapades.

### 3.13 Avatar

Avatar är klassen för den spelarstyrda avataren. Avatar ärver egenskaper från Object, och känner till klassen Projectile. Klassen har två privata variabler, score, och font Klassen har funktionen Fire_projectile, som gör att avataren skjuter.

### 3.14 Projectile

Projectile har en variabel som skiljer om det är spelaren eller en motståndaren som skjuter den. Projectile ärver från Objects och känner till klasserna Avatar och Octupus.

### 3.15 Power-up

Power-up ärver från klassen Objects. Power-up - klassen kommer slumpmässigt skapa en power-up längst up på skärmen. Power-up kommer vid kollision med avataren resultera i att avatarens eller snäckans HP kommer öka med 1.

## 4. Användningsfall/Scenarion

Nedanstående punktlista visar alla de olika användningsfallen/scenarion där olika objekt kan interagera med varandra. För användningsfall mellan states så hänvisar vi till kravspecfikationen.

- Avatarens projektil kan kollidera med bläckfisk.
- Avatarens projektil kan kollidera med hajar.

- Avatarens projektil kan kollidera med snäckorna.
- Bläckfiskens projektil kan kollidera med snäckorna.
- Bläckfiskens projektil kan kollidera med avataren.
- Power-up kan kollidera med avataren.
- Power-down kan kollidera med avataren.
- Hajen kan kollidera med avataren.

Nedan finns ett mer detaljerat exempel av en spelomgång med flera scenarion beskrivna. Konsekvenser av en händelse skrivs med +.

Menyn öppnas -> Spelaren trycker på Highscoreknappen -> Highscorelistan visas -> Spelaren trycker på menyknappen -> Spelaren skriver in sitt namn -> Spelaren bekräftar namnet -> Spelaren startar spelet -> Avataren blir träffad av projektil från bläckfisk + förlorar 1 HP -> Avataren tar en Power-up + får 1 HP -> Avataren avfyrar en projektil som träffar bläckfisk + bläckfisken förvinner -> En snäcka blir träffad av en projektil från en bläckfisk + snäckan förlorar 1 HP. Avataren träffas av haj + förlorar 1 HP -> Avataren träffas av Power-down + rör sig långsammare under en viss tid -> ... -> Avataren har bara 1 HP kvar och träffas av haj + avataren dör -> Spelet avslutas -> Highscorelistan visas.