2. Arkitekturell översikt

2.1 Use-Case View

Bilden visar en överblick på de Use-Cases vi har i vår applikation där Spelaren är vår Actor och LocalDB är den externa aktören. Att kommunikationen mellan spelare och d0atabas fungerar felfritt är absolut prio - annars kan inga spel skapas, laddas eller sparas. Om förbindelsen inte är aktiv är menyn det enda som kommer kunna visas och systemet kommer inte kunna utföra något av valen.

Man kan även se var och vilka Use-Cases som inkluderar varandra i de olika scenariers som finns, men även de som exkluderas (inte är beroende av ett tidigare case).

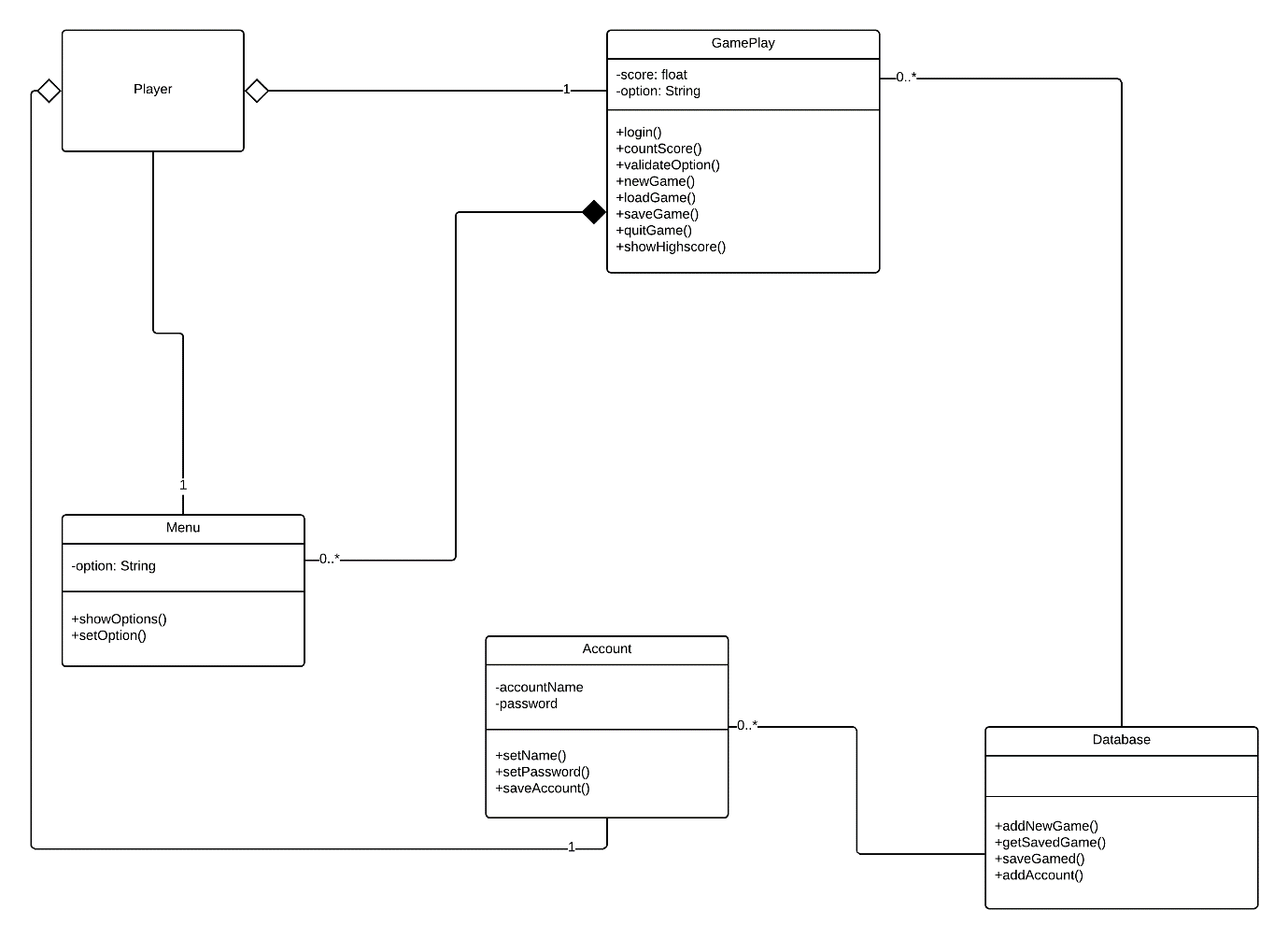
2.2 Use-Case Realization

Här beskrivs några av de Use-Cases som finns tillsammans med tillhörande Story. Ett gemensamt krav som finns är bl.a. att det måste finnas tillräckligt med RAM för att spelet ska fungera korrekt.

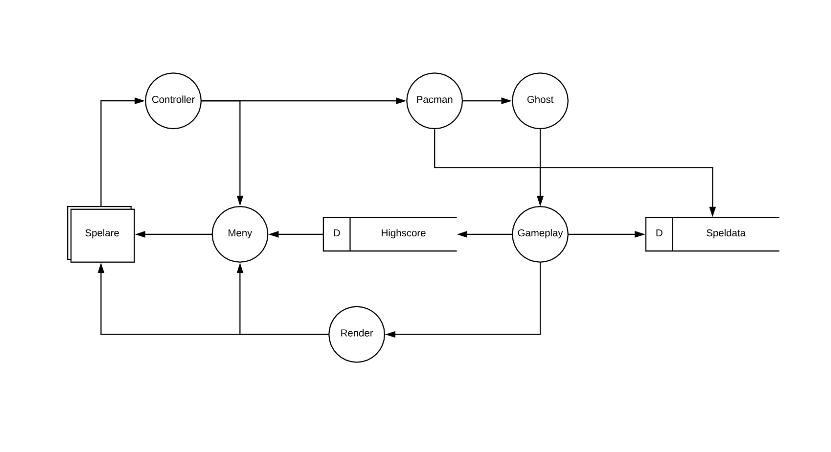
|  |
| --- |
| Use Case: Starta Spelet Actor: Spelaren Description: Användaren startar applikationen och huvudmenyn visas med alternativ att starta spelet, ladda ett befintligt spel eller titta på Highscore-listan. Story: Som användare vill jag komma till en huvudmeny där jag kan göra val när spelet startar. |
| Use Case: Ladda Spel Actor: Spelaren Description: Användaren laddar ett sparat spel från en tidigare session från huvudmenyn. Story: Som användare vill jag kunna ladda spel som jag tidigare har spelat. |
| Use Case: Spara Spel Actor: Spelaren Description: Användaren sparar alla ändringar som har skett efter den senaste sparade sessionen. Story: Som spelare vill jag kunna spara spelet för att ändringar/framsteg inte ska gå förlorade. |

2.3 Logical View

I bilden nedanför illustreras några av de funktioner och klasser som bygger systemet, även relationer dessa emellan. Några klasser är beroende av andra för att de ska kunna existera, medans andra instansieras oavsett.



Nästa bild visar istället hur data skickas mellan de olika objekten/enheterna.

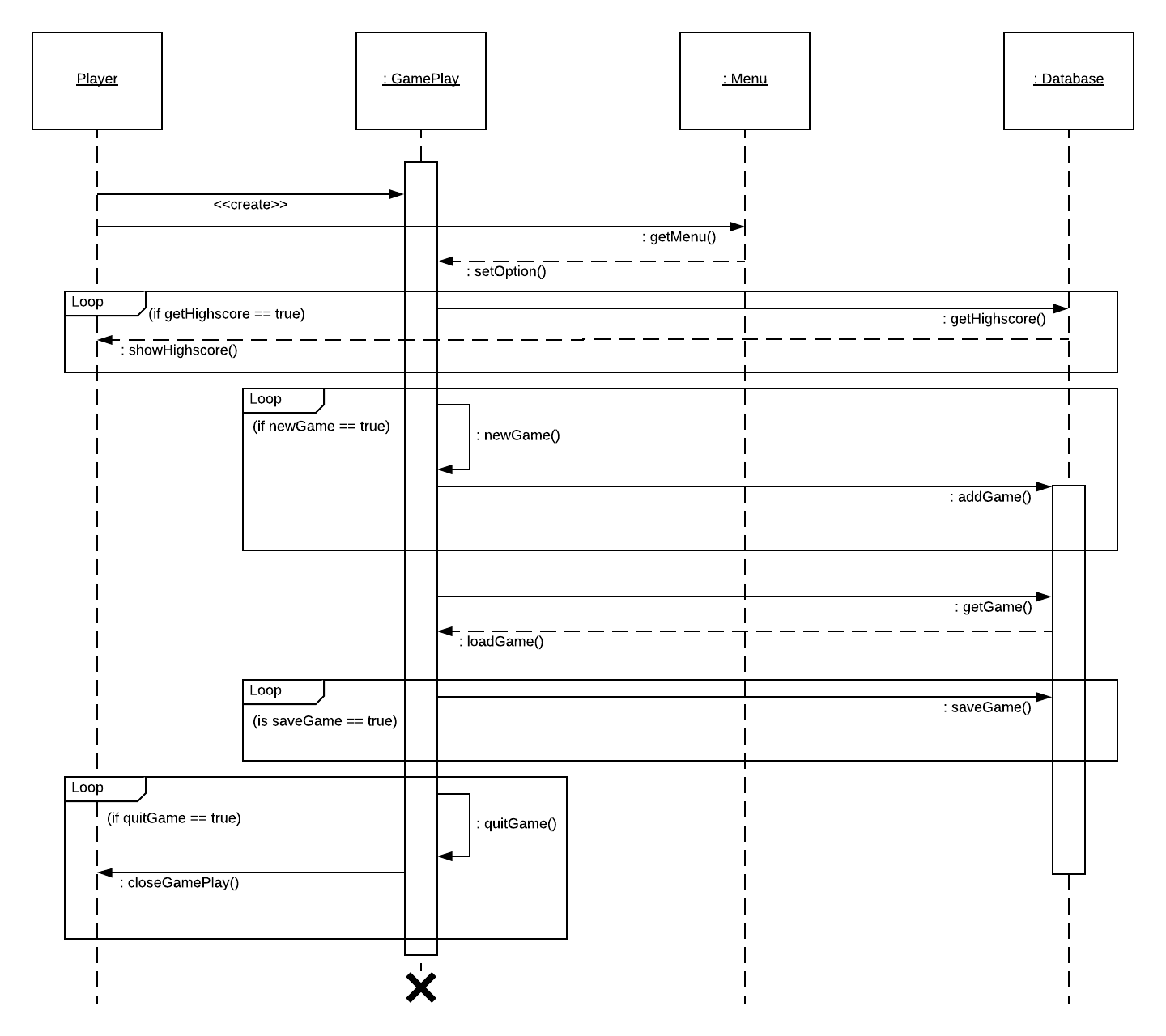


**Översikt av spelets huvudsakliga processer**

|  |  |
| --- | --- |
| **Process** | **Förklaring** |
| Controller | den process som ansvarar för att ta emot input från användaren. |
| Pacman | Uppdaterar spelarens position på spelplanen samt kontrollerar om pacman kan äta spöken eller inte. |
| Ghost | Uppdaterar spökets position på spelplanen. |
| Menu | Den process som låter spelaren göra val att spara, ladda eller avsluta när spelet är pausat eller när användaren är i huvudmenyn. |
| Gameplay | Den process som ansvarar för att räkna poäng, upptäcka kollisioner och ha koll på om spelaren är i livet eller inte. Även den process som kommunicerar med databasen för sparade spel och hämtar dem om användaren begär det. |
| Render | Renderar spelets grafiska komponenter. |

2.4 Interactions

Följande diagram beskriver programmet från det att spelaren har skapat ett konto eller loggat in med befintligt konto. Spelaren kan sedan göra sitt val i menyn och diagrammet visar de funktioner som körs baserat på valet.



* Efter att GamePlay har skapats så kallas funktionen getMenu() som presenterar de olika valen som finns.
* När spelaren har gjort sitt val så skickas detta in i GamePlay där valet valideras och tilldelas med setOption().
* Sedan följer olika loopar/funktioner som körs beroende av spelarens val:  
  - getHighscore() hämtar data från databasen som sedan visas för spelaren.  
  - newGame() skapar ett nytt spel som sedan läggs till i databasen.  
  - loadGame() körs när ett sparat spel har hittats och hämtats från databasen.  
  - saveGame() sparar den aktuella sessionen, skriver över tidigare version i databasen.  
  - quitGame() avslutar GamePlay och förstör även det objektet.