TDDE30 projekt: Tower Defence

Introduktion till projektet

Under projektet kommer jag att utveckla ett Tower Defence-spel där spelaren placerar försvarstorn för att stoppa vågor av fiender. Spelet kommer att inkludera olika typer av torn med unika egenskaper, uppgraderingsmöjligheter samt en balanserad progression av svårighetsgrad. Fokus ligger på strategiskt tänkande och spelbarhet.

Bakgrund

Tower Defence (TD) är en genre av strategispel där spelaren ska försvara en bas, en väg eller ett område från inkommande fiender genom att strategiskt placera ut olika försvarstorn. Fienderna rör sig oftast längs en förutbestämd väg och tornens uppgift är att skjuta eller på annat sätt påverka dem för att förhindra att de når sitt mål.

Grundläggande mekanik

- **Bana och fiender**: Spelvärlden innehåller en väg eller flera vägar där fiender rör sig från en startpunkt till en slutpunkt. Fienderna kan ha olika hastigheter, liv och specialförmågor.
- **Torn och uppgraderingar**: Spelaren kan placera ut torn på specifika platser eller fritt inom vissa begränsningar. Torn kan ha olika egenskaper som räckvidd, skada, skjut frekvens och specialeffekter. De kan också uppgraderas för att bli mer kraftfulla.
- **Resurshantering**: Spelaren får resurser genom att besegra fiender eller med tiden. Dessa resurser används för att köpa och uppgradera torn.
- **Svårighetsgrad och vågor**: Fienderna kommer i vågor med ökande svårighetsgrad. Senare vågor kan introducera nya typer av fiender med särskilda egenskaper.

Se Wikipedias artikel för mer information https://en.wikipedia.org/wiki/Tower_defense även

TDDE30
Projekt: Tower Defence

Milstolpar

För att bygga mitt Tower Defense spel kommer jag använda flera milstolpar med under mål för att kunna jobba effektivt med utvecklingen

Milstolpar för utvecklingen

Steg	
	Grundläggande funktionalitet
1	Skapa en spelplan med en grundläggande visuell representation av en bana.
2	Implementera fiender som kan röra sig längs en fördefinierad bana.
3	Skapa ett grundläggande torn som kan placeras på vissa platser.
4	Implementera enkel skottmekanik så att torn kan skjuta på fiender inom räckhåll.
5	Lägga till enkel kollision och fiender kan ta skada och elimineras vid 0 HP.
6	Implementera grundläggande resurssystem där spelaren tjänar poäng för besegrade fiender.
7	Förlust vilkor för att avsluta spelet
	Utveckling av spelets mekanik
8	Lägga till möjlighet att uppgradera torn för ökad skada eller räckvidd.
9	Implementera olika typer av fiender med varierande hastighet och HP.
10	Skapa ett enkelt användargränssnitt som visar resurser och aktuell våg.
11	Implementera vågmekanik där fiender kommer i fördefinierade vågor.
	Förbättrad spelupplevelse
12	Implementera fler torn med olika egenskaper, såsom långsammande eller spridningsskott.
13	Implementera dynamiska vågor där svårighetsgraden ökar över tid.
14	Skapa en meny med start- och pausfunktioner.
15	Implementera ljud- och ljudeffekter för skott och fiender.
16	Förbättra grafik och animationer.
17	Flera banor
18	Paus funktion
19	Förbättrade projektiler med varierande hastighet och träffsäkerhet
20	Rivning av torn

	Avancerade funktioner och extra innehåll
17	Implementera en kartredigerare där spelaren kan designa egna banor.
18	Implementera stöd för olika spellägen, t.ex. oändligt läge.
19	Lägga till power-ups eller specialförmågor för spelaren.
20	Implementera ett sparsystem för högsta poäng och sparade framsteg.

Övriga implementation förberedelser

Jag planerar att använda en uppdelning av spelets funktionalitet i flera klasser för att skapa en strukturerad och lättöverskådlig kodbas. Följande klasser kommer att användas:

- **GameManager** – Hanterar spelets övergripande logik och styr spelets flöde.
- **Map** – Representerar spelplanen och hanterar information om kartan.
- **Enemy** – Representerar fiender i spelet och deras beteende.
- **Projectile** – Hanterar projektiler som avfyras av torn och deras interaktion med fiender.
- **WaveManager** – Ansvarar för att hantera fiende spawns och vågor av fiender.
- **Viewer** – Sköter rendering och UI.
- **SaveManager** – Ansvarar för att spara och ladda speldata.

För att säkerställa en flexibel och skalbar kodbas kommer jag att använda följande designmönster:

- **Observer Pattern** – Används för att uppdatera UI när resurser ändras, vilket möjliggör en reaktiv användarupplevelse.
- **State Pattern** – Hanterar olika speltillstånd såsom meny, spel, paus och game over.
- **Factory Pattern** – Används för att dynamiskt skapa fiender och torn, vilket underlättar vid framtida utbyggnad av spelet.

Spelets rendering och UI kommer att implementeras med **JFrame**, medan **Gson** används för att spara och läsa in speldata. Gson kommer även att användas vid laddning av kartfiler i ett senare skede.