Projektbeskrivning

Tower Defense

2042-03-22

Projektmedlemmar:
Axel Björk - axebj824@student.liu.se
Wilhelm Malé - wilma870@student.liu.se

Handledare:
Morgan Nordberg - morno368@student.liu.se

Innehåll

Projektplan

1. Introduktion till projektet

Detta projekt inspireras av inspirationsprojektet Tower Defence.

Tower Defence är en kategori av spel med rötter till 1980-talet som går ut på att försvara en punkt (ofta symboliserat som en bas eller by) från horder av monster som går genom en bana. Detta görs genom att bygga torn som skjuter ner dem. Likt många andra spelkategorier finns det olika tappningar men man kan göra en grov uppdelning i två kategorier:

- Monstren går i en smal bana och spelarna bygger torn utanför banan.
- Monstren går i en bredare bana (kan vara hela spelplanen) och spelarna bygger torn inuti banan. I de här versionerna brukar tornen blockera motståndarna och/eller skjuta på dem.

2. Ytterligare bakgrundsinformation

Se Wikipedias artikel: https://en.wikipedia.org/wiki/Tower_defense.

3. Milstolpar

#	Beskrivning
1	Det finns datatyper för kvadrater som man använder för att bygga upp en karta som under senare steg ska gå att placera ut fiender och försvar på. Kvadraterna ska vara en enumtyp som representerar t.ex. väg, placerbart, torn, fiender, bas. Även ett fönster för att kunna se kartan ska implementeras här.
2	Fiender ska röra sig längs med på kartan vägen när spelet startas.
3	Minst en typ torn ska kunna placeras ut på designerade platser i kartan.
4	Basen ska kunna förstöras och man ska kunna förlora ifall en fiende tar sig dit.
5	Tornen ska kunna skjuta eller attackera fiender så att dem försvinner. Detta ska fungera med en specifik radie.
6	Vågor av fiender ska implementeras. För ökande svårighet under spelets gång.
7	Spelet är fungerande. (Oändlig till förlust)

8	HP på Fiender så att de ej dör efter 1 skott. Basen ska även ha HP. Fiender ska stanna vid basen (om de hinner dit) medan spelet är i gång.
9	Man ska kunna få pengar av att döda fiender för att kunna placera ut tornen
10	Det ska gå att markera ett torn, och få upp en meny för att kunna ta bort tornet. Menyn ska vara inom spelets fönster, inte något eget.
11	Menyn ska utökas till att man ska kunna uppgradera tornet på minst 1 sätt.
12	Lägger till ett highscore som uppdateras när fiender är dödade. Ska även visa vilken våg man är på.
13	Paus i spelet. Allt ska stoppas medan det är pausat.
14	Lägg till en ”boss” vid någon specifik våg och minst en till fiendetyp (snabb, långsam men mycket hp) som skiljer den från fienderna innan.
15	Lägger till 2 till typer av torn och alla torn har minst 2 uppgraderingar.
16	Ljudeffekter och Sprites för torn, fiender, bas, placerbar tile, etc.
17	Lägg till en startmeny för att starta spelet, med simpla inställningar som exempelvis att ändra volymen i spelet.
18	Utöka menyn för att välja en annan karta. Här måste då alltså en till karta göras och lagras.
19	Skapa en shop där man får köpa föremål som spelaren kan använda för att få globala effekter, atombomb som tar bort alla fiender på kartan, eller att alla torn får 1m attackhastighetshöjning.
20	Torn med effekter som t.ex. damage over time, saktar ner fiender, påverkan på andra torn.
21	Prioritering av fiender att sikta på för torn. Användaren ska ha möjlighet att välja om tornet ska skjuta på den första fienden inom dennes radie eller fienden med mest maximala HP.
22	Utöka animationer för att skjuta på fiender.
23	Lägga till flera torn, fiender, uppgraderingar, globala effekter.

4. Övriga implementationsförberedelser

Kvadrat Enumtype som används för att skapa kartan, fiender och torn.

Abstractclass entity för torn och fiende. Fält i denna är: position, storlekt, damage, attackspeed, färg/sprite,

Torn kräver position, storlek, radie, attackspeed, damage, färg/sprite, attackmetod (senare: prioritering, upgraderingsmetod, säljmetod, mm).

Fiende kräver position, storlek, HP, speed, damage, attackspeed, färg/sprite, Förflyttningsmetod, attackmetod.

En klass för **karta**, temporär hårdkodad och en klass för att läsa och skapa en karta från en JSON fil.

Bas Som försvaras kräver endast HP

Board som sätterihop karta, torn och fiender.

BoardViewer som ritar ut i en JFrame. JFrame baserad på kartans storlek

GameState som kör timer för att uppdatera spelet. Och andra saker som är generella för spelet som t.ex. Highscore, våg.

UpgradeMenu som är menu som tar in ett torn och utifrån det kan kalla på tornets egna funktioner för att uppgradera sig eller ta bort sig själv. Den tar även in pengar och håller koll på logiken där.

Shop som är shopen för globala items. Tar också in pengar.

MainMenu klass för mainmenyn som säger åt klassen **karta** vilken karta som ska laddas in och initierar board när man vill köra spelet.

Pausebutton pausknapp som pausar spelet. Måste kalla på funktion i gamestate som stoppar timern.

5. Utveckling och samarbete

Vi har båda samma ambitionsnivå och siktar på en 5:a. Vi vill båda bli klara till deadline.

Vi kommer att jobba på alla schemalagda tillfällena men inte nödvändigtvis på plats. Men kommer även att jobba utanför schemalagd tid både tillsammans och enskilt.

Vi sätter ingen hård gräns på milstolpar men vill vara klara med så mycket som möjligt och rapporten den 24/03/25 vilket är deadlinen. Vi sätter deadlines på milstolpar ifall vi känner att det går långsamt.

Om man är såpass sjuk att man inte kan jobba hemifrån eller om man har något bokat sjukvårdsbesök eller liknande.

När vi arbetar tillsammans kommer vi alltid att använda Liveshare. Vi kommer i vissa fall planera att arbeta på delar enskilt. Vi kommer att försöka en gång i veckan gå igenom kodanalysen och rätta varningar och även förklara vad all kod gör för varandra.

(Resten av dokumentet ska inte lämnas in förrän projektet är klart, men titta ändå genom allt för att se vilka delar ni behöver arbeta med och fylla i *kontinuerligt* under projektets gång!)

Projektrapport

Även om denna del inte ska lämnas in förrän projektet är klart, är det **viktigt att arbeta med den kontinuerligt** under projektets gång! Speciellt finns det några avsnitt där ni ska beskriva information som ni lätt kan glömma av när veckorna går (vilket flera tidigare studenter också har kommenterat).

Tänk på att ligga på lagom ambitionsnivå! En välskriven implementationsbeskrivning (avsnitt 6) hamnar normalt på **3-6 sidor** i det givna formatet och radavståndet, med ett par mindre UML-diagram och kanske ett par andra små illustrerande bilder vid behov. Hela projektrapporten (denna sista halva av dokumentet) behöver sällan mer än 10-12 sidor.

6. Implementationsbeskrivning

I det här avsnittet, och dess underavsnitt (6.x), beskriver ni olika aspekter av själva *implementationen*, under förutsättning att läsaren redan förstår vad *syftet* med projektet är (det har ju beskrivits tidigare).

Tänk er att någon ska vidareutveckla projektet, kanske genom att fixa eventuella buggar eller skapa utökningar. Då finns det en hel del som den personen kan behöva förstå så att man vet *var* funktionaliteten finns, *hur* den är uppdelad, och så vidare. Algoritmer och övergripande design passar också in i det här kapitlet.

Bilder, flödesdiagram, osv. är starkt rekommenderat!

Skapa gärna egna delkapitel för enskilda delar, om det underlättar. **Ta inte bort några rubriker!**

Även detta är en del av examinationen som visar att ni förstår vad ni gör!

6.1. Milstolpar

Ange för varje milstolpe om ni har genomfört den helt, delvis eller inte alls.

Detta är till för att labbhandledaren ska veta vilken funktionalitet man kan ”leta efter” i koden. Själva bedömningen beror *inte* på antalet milstolpar i sig, och inte heller på om man ”hann med” milstolparna eller inte!

6.2. Dokumentation för programstruktur, med UML-diagram

Programkod behöver dokumenteras för att man ska förstå hur den fungerar och hur allt hänger ihop. Vissa typer av dokumentation är direkt relaterad till ett enda fält, en enda metod eller en enda klass och placeras då lämpligast vid fältet, metoden eller klassen i en Javadoc-kommentar, *inte här*. Då är det både enklare att hitta dokumentationen och större chans att den faktiskt uppdateras när det sker ändringar. Annan dokumentation är mer övergripande och saknar en naturlig plats i koden. Då kan den placeras här. Det kan gälla till exempel:

- **Övergripande programstruktur**, t.ex. att man har implementerat ett spel

som styrs av timer-tick n gånger per sekund där man vid varje sådant tick först tar hand om input och gör eventuella förflyttningar för objekt av typ X, Y och Z, därefter kontrollerar kollisioner vilket sker med hjälp av klass W, och till slut uppdaterar skärmen.

- **Översikter över relaterade klasser** och hur de hänger ihop.
    - Här kan det ofta vara bra att använda **UML-diagram** för att illustrera – det finns även i betygskraven. Fundera då först på vilka grupper av klasser det är ni vill beskriva, och skapa sedan ett UML-diagram för varje grupp av klasser.
    - Notera att det sällan är särskilt användbart att lägga in hela projektet i ett enda gigantiskt diagram (vad är det då man fokuserar på?). Hitta intressanta delstrukturer och visa dem. Ni behöver normalt inte ha med fält eller metoder i diagrammen.
    - **Skriv sedan en textbeskrivning av vad det är ni illustrerar med UML-diagrammet**. Texten är den huvudsakliga dokumentationen medan UML-diagrammet hjälper läsaren att förstå texten och få en översikt.
    - IDEA kan hjälpa till att göra klassdiagram som ni sedan kan klippa och klistra in i dokumentet. Högerklicka i en editor och välj Diagrams / Show Diagram. Ni kan sedan lägga till och ta bort klasser med högerklicksmenyn. Exportera till bildfil med högerklick / Export to File.

I det här avsnittet har ni också en möjlighet att visa upp era kunskapen genom att diskutera koden i objektorienterade termer. Ni kan till exempel diskutera hur ni använder och har nytta av (åtminstone en del av) objekt/klasser, konstruktorer, typhierarkier, interface, ärvning, overriding, abstrakta klasser, subtypspolymorfism, och inkapsling (accessnivåer).

Labbhandledaren och examinatorn kommer bland annat att använda dokumentationen i det här avsnittet för att förstå programmet vid bedömningen. Ni kan också tänka er att ni själva ska vidareutveckla projektet efter att en annan grupp har utvecklat grunden. Vad skulle ni själva vilja veta i det läget?

När ni pratar om klasser och metoder ska deras namn anges tydligt (inte bara ”vår timerklass” eller ”utritningsmetoden”).

Framhäv gärna det ni själva tycker är **bra/intressanta lösningar** eller annat som handledaren borde titta på vid den senare genomgången av programkoden.

Vi räknar med att de flesta projekt behöver runt **3-6 sidor** för det här avsnittet.

7. Användarmanual

När ni har implementerat ett program krävs det också en manual som förklarar hur programmet fungerar. Ni ska beskriva programmet tillräckligt mycket för att en labbhandledare själv ska kunna *starta det, testa det och förstå hur det används*.

Inkludera flera (**minst 3**) **skärmdumpar** som visar hur programmet ser ut! Dessa ska vara ”inline” i detta dokument, inte i separata filer. Sikta på att visa de relevanta delarna av programmet för någon som *inte* startar det själv, utan bara läser manualen!

(Glöm inte att ta bort våra instruktioner, och exportera till PDF-format med korrekt namn enligt websidorna, innan ni skickar in!)