Projektbeskrivning

<ÖÖ Tower defence>

2021-03-10

Projektmedlemmar:
Jakob Österberg <jakos322@student.liu.se>
Anton Östman <antos931@student.liu.se>

Handledare:
Piotr Rudol < piotr.rudol@liu.se >

1. Introduktion till projektet 2
2 Ytterligare bakgrundsinformation 2
3 Milstolpar 2
4. Övriga implementationsförberedelser 4
5. Samarbete 5
6. Implementationsbeskrivning 6
6.1. Milstolpar 6
6.2. Dokumentation för programstruktur, med UML-diagram 7
6.2.1 Spelets karta, klasserna GameMap och Tile 7
6.2.2 Klasserna Entity, EntityAttacker, Tower, Enemy och Projectiles 8
6.2.3 Spellogiken och klassen GameHandler 11
6.2.4 Start av spel, användargränssnitt och grafik 12
7. Användarmanual 13

Projektplan

1. Introduktion till projektet

Motståndare går längs en väg från ena sidan av skärmen till den andra. Spelarens mål är att förhindra det genom att bygga ett försvar med torn som är placerade på strategiska positioner bredvid vägen. (Se bild 1.1 och 1.2)

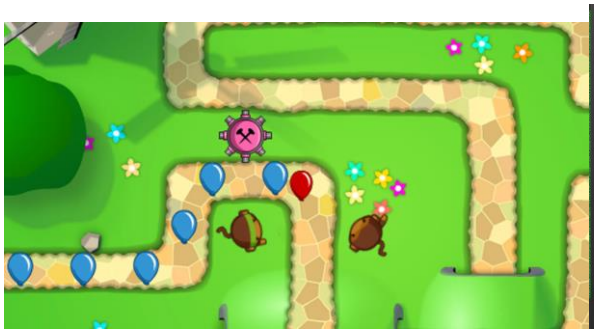

Bild 1.1

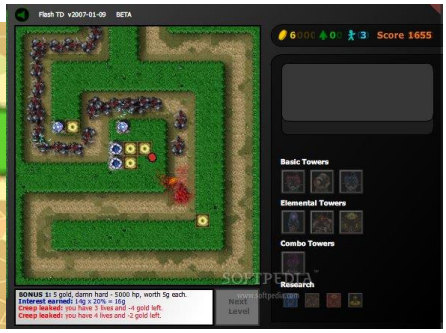

Bild 1.2

Inspirationsprojektet “Tower Defence” har använts för att skapa en grundlig bild av hur projektet kan påbörjas.

2 Ytterligare bakgrundsinformation

Regler för spelet:
Fiender kan bara gå längs vägen från start till slut.
Torn kan bara placeras bredvid vägen på gräsblock.
Om fiender kommer till slutet av vägen tappar spelaren liv.
Då spelaren får slut på liv är spelet över.
Torn kostar pengar, pengar fås genom att besegra fiender och vid starten av spelet

3 Milstolpar

Den preliminära färdplanen som projektet planeras ta illustreras med bilden nedan (se bild 3.1.). De första stegen har störst relevans medan de senare eventuellt ändras under projektets gång.

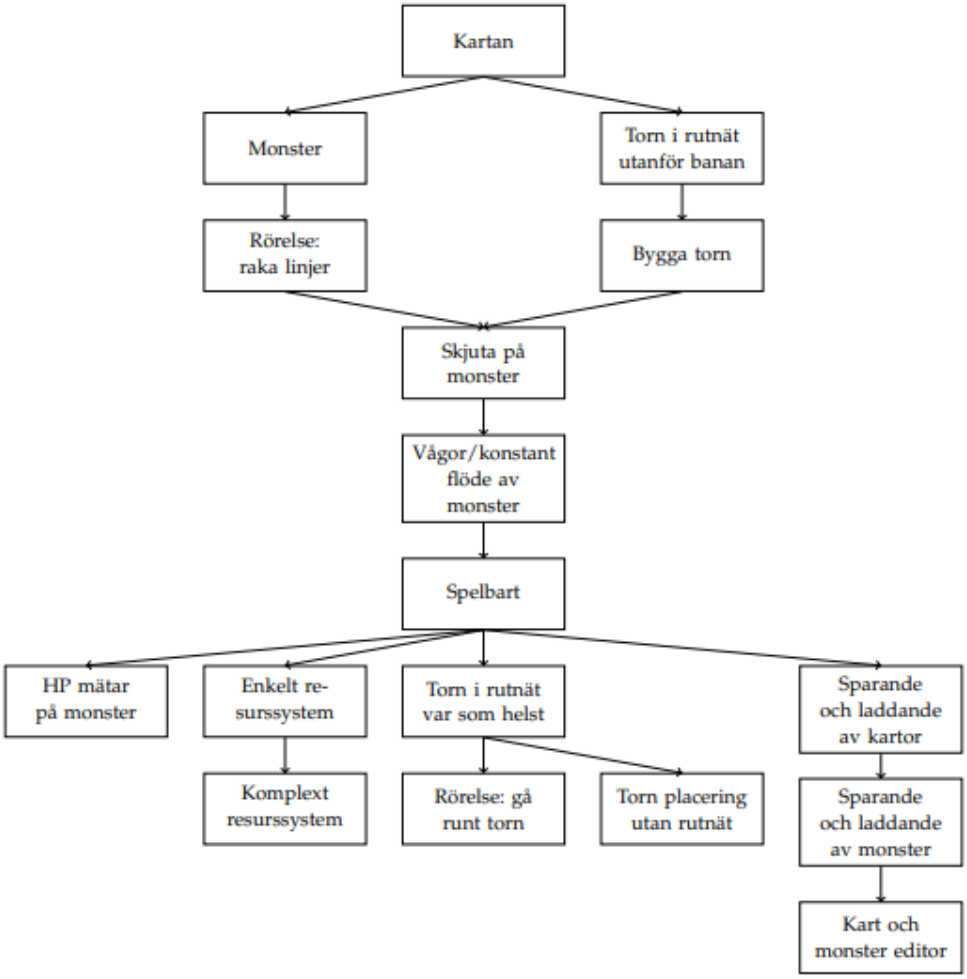

Bild 3.1 (Från "Tower Defence" inspirations projektet.)

De olika stegen i projektet planeras att tas steg för steg tills att spelet är spelbart. När spelet är spelbart kan eventuell vidareutveckling göras i godtycklig ordning. Se tabell 3.2

Tabell 3.2 med de preliminära målen i projektet.

1	Skapa klassen GameMap, Implementera alla typer av block som kan vara på kartan.
2	Hårdkodad karta.
3	Skapa klassen mapViewer som visar kartan och dess olika block i olika färger.
4	Skapa en Entity klass/interface/abstract klass / olika datatyper, som grund till monster och torn.
5	Torn.
5.1	Placera ett “tomt” torn på kartan som är synligt på kartan, ingen funktionalitet.
5.2	Skapa olika torntyper som alla ärver av en gemensam tornklass

5.3	Implementera funktioner för alla torn Planerad funktionalitet: Torn som skadar enskilda fiender. Torn som skadar flera fiender med ett skott likt en explosion Torn med projektiler som penetrerar fiender och därmed kan skada alla fiender som står efter varandra i en rad.
6	Implementera motståndare.
6.1	Fiender ska kunna läggas till på kartan.
6.2	Rörelse av fienderna.
6.3	Fiender ska ha liv ha liv och en träffruta som torn ska träffa för att skada dem.
6.4	Fiender ska komma in eftersom i vågor.
7	Pengar system.
7.1	Pengar ska fås vid start av spel och vid start av en våg.
7.2	Fiender som besegras ska ge pengar.
7.3	Skapa torn som ger pengar till spelaren.
8	Användargränssnitt.
8.1	Spelaren pengar ska visas.
8.2	Torn ska kunna köpas i en meny och läggas ut på spelplanen.
8.3	Kostnad på torn ska synas.
	Spelet är SPELBART eventuell vidareutveckling nedan. Kan göras i obestämd ordning.
9	Spelaren ska kunna välja bland olika kartor att spela på
10	Lägga till ett startgränssnitt med olika alternativ användaren kan välja innan start.
10.1	Användaren ska kunna välja bland olika kartor.
10.2	Välja bland olika svårighetsgrader.
10.3	Lägga till en poängtavla där de bästa spelomgångarna syns.
11	Ökande svårighetsgrad på fiender.
12	En egen kartskapare där spelaren kan skapa egna kartor.
13	Uppgraderingar på torn.
13.1	Torn ska kunna uppgraderas med mer skada, räckvidd och skjuthastighet.
13.2	Uppgradera via en knapp i menyn.
13.3	Speciella uppgraderingar som ändrar inriktning på tornet. Exempelvis ett torn som i början bara skjuter en projektil kan uppgraderas till att skjuta två samtidigt.
14	Lägga till fler torn i spelet Exempelvis torn som saktar ner fienden.
15	Tillägg på fiender.
15.1	En extra stark fiende men annorlunda utseende som dyker upp var tionde våg eller liknande.
15.2	Fiender med olika immuniteten som bara kan ta skada av specifika torn.
16	Möjlighet att ladda in olika bilder som exempelvis PNG format för att byta utseendet på fiender eller torn.

4. Övriga implementationsförberedelser

Vissa delar av den generella strukturen för programmet kan planeras i förväg, och den informationen återfinns under denna rubrik.

Användargränssnitt kopplat till towerDefenceComponent som ritar ut kartan samt de objekt som finns på kartan. Objekten vet själv hur de ritas ut och towerDefenceComponent ska inte ha inbyggd logik för hur de enskilda objekten ska ritas ut.

Klassen för kartan är den enda som får och kan manipulera sin egen struktur. Om andra klasser måste ändra något måste den gå direkt via kartan först.
Klassen Entity planeras vara ett gränssnitt som implementeras av de abstrakta klasserna Tower och Enemy. Se bild 4.1 för diagram på planerad koppling mellan de olika klasserna

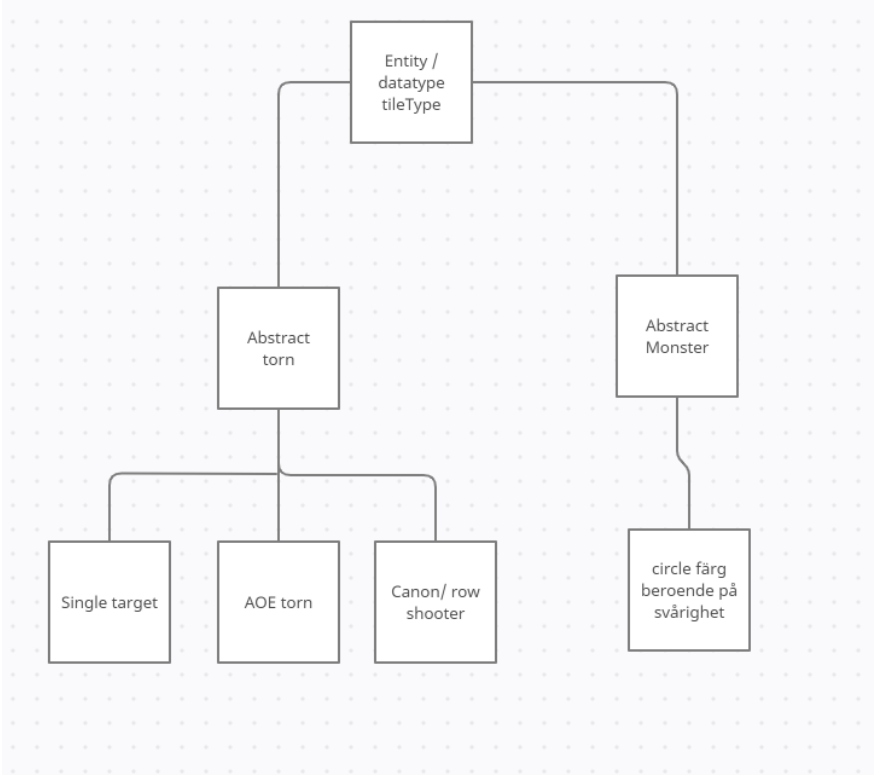

Bild 4.1

5. Samarbete

Samma ambitionsnivå båda projektmedlemmar siktar på betyg 5.
Arbetstider: Som minst används labbtillfällena för att arbeta tillsammans övrig tid är eget arbete om inte annat bestäms.
Den preliminära tidslinjen avser tiden från början av april till slutet av maj.

50% av tiden läggs på att få spelet fungerande.
35% av tiden används till ytterligare utökning av spelet.
15% av tiden spenderas för att finslipa koden.

6. Implementationsbeskrivning

Implementationsbeskrivningen förklarar de olika delarna av programkoden. Avsikten är att ge en översiktlig bild av koden samt vara ett referensdokument till eventuell framtida utveckling av programmet.

6.1. Milstolpar

Milstolparna i projektet ska ge konkreta delmål för att det alltid finnas ett mål som går att arbeta mot. Det finns även som referens för vad som kan förväntas beskrivas i avsnitt 6.2. Se tabell 6.1.1 för milstolparna

Tabell 6.1.1

De planerade stegen i projektet som utförs ett steg i taget fram tills vidareutveckling.
Grön markering innebär implementerad, gult innebär delvis implementerat och röd innebär inte implementerat.

1	Skapa klassen GameMap, Implementera alla typer av block som kan vara på kartan
2	Hårdkodad karta
3	Skapa klassen mapViewer som visar kartan och dess olika block i olika färger
4	Skapa en entity klass/interface/abstract klass / olika datatyper, som grund till monster och torn
5	Torn
5.1	Placera ett “tomt” torn på kartan som går att se, ingen funktionalitet.
5.2	Skapa olika torntyper som alla ärver av en gemensam tornklass
5.3	Implementera funktioner för alla torn Planerad funktionalitet: Torn som skadar enskilda fiender. Torn som skadar flera fiender med ett skott likt en explosion Torn med projektiler som penetrerar fiender och därmed kan skada alla fiender som står efter varandra i en rad.
6	Implementera motståndare
6.1	Fiender ska kunna läggas till på kartan
6.2	Rörelse av fienderna
6.3	Fiender ska ha liv och en träffruta som torn ska träffa för att skada dem
6.4	Fiender ska gradvis komma in i kartan som vågor
7	Pengar system
7.1	Pengar ska fås vid start av spel och vid start av en våg
7.2	Fiender som besegras ska ge pengar
7.3	Skapa torn som ger pengar till spelaren
8	Användargränssnitt
8.1	Spelarens pengar ska visas
8.2	Torn ska kunna köpas i en meny och läggas ut på spelplanen

8.3	Kostnad på torn ska synas
	Spelet är SPELBART eventuell vidareutveckling nedan. Vidareutvecklingen kan ske i obestämd ordning
9	Spelaren ska kunna välja bland olika kartor att spela på
10	Lägga till ett startgränssnitt med olika alternativ användaren kan välja innan start
10.1	Användaren ska kunna välja bland olika kartor
10.2	Välja bland olika svårighetsgrader
10.3	Lägga till en poängtavla där de bästa spelomgångarna syns
11	Ökande svårighetsgrad på fiender
12	En egen kartskapare där spelaren kan skapa egna kartor
13	Uppgraderingar på torn
13.1	Torn ska kunna uppgraderas med mer skada, räckvidd och skjuthastighet
13.2	Uppgradera via en knapp i menyn
13.3	Speciella uppgraderingar som ändrar inriktning på tornet. Exempelvis ett torn som i början bara skjuter enskilda fiender kan uppgraderas till att skjuta två samtidigt
14	Lägga till fler torn i spelet. Exempelvis torn som saktar ner fienden
15	Tillägg på fiender
15.1	En extra stark fiende men annorlunda utseende som dyker upp var tionde våg eller liknande.
15.2	Fiender med olika immuniteten som bara kan ta skada av specifika torn.
16	Möjlighet att ladda in olika bilder som exempelvis PNG format för att byta utseendet på fiender eller torn.

6.2. Dokumentation för programstruktur, med UML-diagram

Den huvudsakliga dokumentationen för programmets struktur och hur de olika klasserna interagerar beskrivs under rubrik 6.2.

6.2.1 Spelets karta, klasserna GameMap och Tile

Spelplanen är en av de grundläggande delarna i spelet. Spelplanen är uppbyggd av ett rutnät av block som heter Tile.

Klassen Tile innehåller information om vilken typ av spelblock, färg samt position i rutnätet som bygger upp spelplanen. Spelplanen består av två olika typer av Tile objekt som representeras av Enum-typerna ROAD och GRASS, som härefter benämnas som vägblock och gräsblock. Spelet är uppbyggt sådant att fienden rör sig över vägblocken medan torn placeras på gräsblock.

Uppbyggnaden av spelplanen kan variera beroende på vilken karta som väljs att spela på. Kartor lagras som MapInfo objekt som sparar både kartans storlek samt motståndanas väg. Detta innebär att när maps.json filen med de sparade kartorna laddas, kan kartorna laddas som MapInfo-objekt. Detta gör att alla kartor kan lagras i en lista samt att informationen om kartan kan fås genom att anropa MapInfo-objektet.

Maps.json filen hanteras med hjälp av java resurser, det innebär att maps.json laddas in med de resterande klasserna och kan gömmas från användaren i en .jar fil. För att

editera kartornas information kan maps.json direkt manipuleras. Exempelvis för att ändra dimensioner på kartan eller lägga till ytterligare vägblock som fienden ska gå på för att ta sig till målet. Vägblock kan inte vara utanför dimensionerna för kartan, då krashar spelet med ett IndexOutofbounds exception när filen laddas.

I centrum av spelplanen är klassen GameMap. Det är denna klass som båda kan ladda in filen med de tillgängliga kartorna samt att ladda den karta som användaren har valt vid startmenyn (se 6.2.4). Detta innebär att GameMap har informationen om hur spelplanen ser ut samt vilken väg fienden ska gå.

6.2.2 Klasserna Entity, EntityAttacker, Tower, Enemy och Projectiles

En central del i ett tower-defence spel är att olika typer av objekt kan interagera på en spelplan. De objekt som kan befinna sig på spelplanen är fiender, torn och projektiler. Dessa typer av objekt delar flera egenskaper och därmed skapades den abstrakta klassen Entity samt de tre abstrakta subklasserna Enemy, Tower och projectiles.

Entity innehåller fält som är gemensamma för det som kan placeras på spelplanen, exempelvis position, färg och storlek. Entity har även metoder som är gemensamma för de objekt som befinner sig i spelet.

En av dessa är move() som innehåller logik för hur en Entity rör sig mot den valda Point2D movePosition som är ett fält. Subklasserna Projectiles och Enemy utökar move() med extra logik. Det innebär att klasserna skriver över move() och använder sig av super-metoden för att röra sig till olika punkter beroende på diverse villkor som klasserna Projectiles respektive Enemy själva definierar.

Entity objekt använder sig av Point2D.Double koordinater för att representera vart de befinner sig i spelet. Det innebär att Entity objekt kan ritas ut och hanteras som att de är mellan två Tile objekt i gameMap (se 6.2.1) på grund av att x-, y-koordinaterna kan vara av typen Double.

Utöver Entity finns även klassen EntityAttacker (se 6.2.2.1). En EntityAttacker är en typ av Entity med utökad logik för att kunna attackera andra Entity objekt. Tower och Projectiles ärver från denna klass då de behöver möjligheten att försvara mot fienden genom att attackera Entity-objektet Enemy.

Inom EntityAttacker defineras metoder som bestämmer när och hur en EntityAttacker attackerar, de viktigaste metoderna är:
canAttack(Entity) som dels använder sig av Entity metoden canBeAttacked() för att bestämma om en EntityAttacker kan attackera en Entity.
DecideEnemy(Entity) som anger logiken för hur enskilda EntityAttacker objekt ska bestämma vilket Entity objekt de ska attackera. Exempelvis om en lista av fiender loopas igenom används decideEnemy(Entity) på alla fiender enskilt för att välja vilken som ska attackeras. Projectile och Tower skriver båda över de förenämnda metoderna och tillsätter ytterligare egen logik utöver det som finns i EntityAttacker.

Det betyder att klasserna som ärver från EntityAttacker kan hantera attacklogik med hjälp av polymorfism. Alla typer av Entity objekt kan bli attackerade av EntityAttacker objekt. Alltså kan exempelvis i en vidareutveckling av programmet projektiler skada

torn om de missar Enemy objektet. Då skulle det kräva av spelaren att placera tornen med varsamhet.

Det finns dessutom en till klass EntityFactory som innehåller statiska metoder för att returnera nya Tower,Enemy och Projectile objekt. Då används enum typerna TowerType, EnemyType och ProjectileType som representerar de specifika objekten. För att hämta ett Entity objekt skrivs exempelvis EntityFactory.*getTower*(TowerType.CANON) för att returnera ett nytt CanonTower objekt.

De abstrakta klasserna Tower och Projectiles som ärver från EntityAttacker och den abstrakta klassen Enemy som ärver av Entity finns i seperata paket där även deras subklasser finns.

Paketens innehåll diskuteras utförligare nedan.

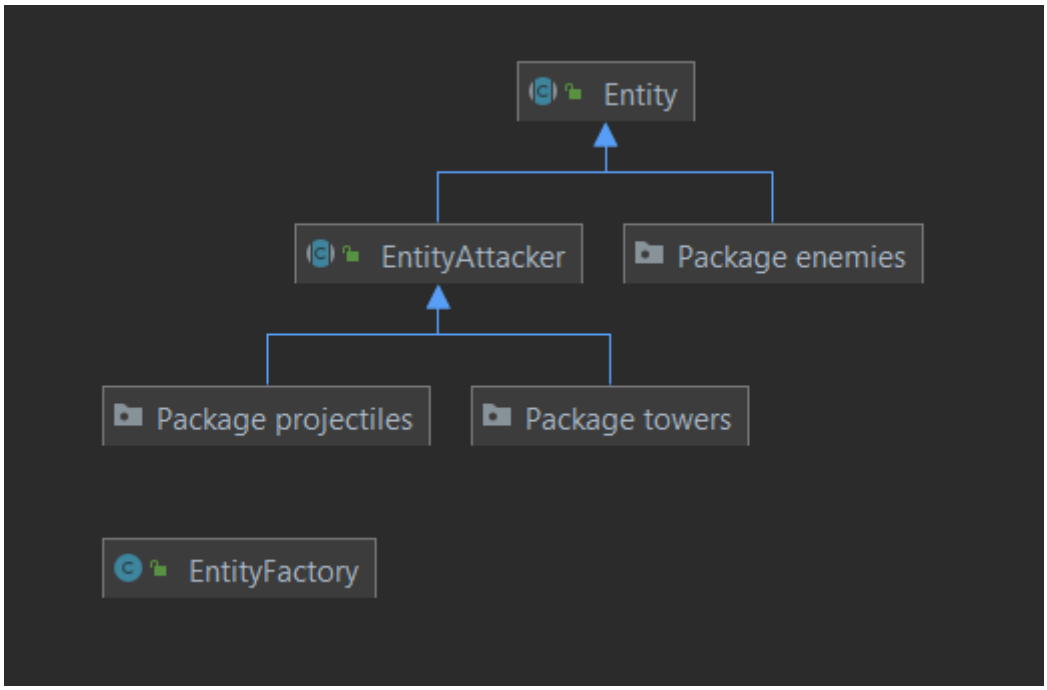

Bild 6.2.2.1

Den abstrakta klassen Tower representerar de objekt som spelaren ska använda sig av för att besegra fienden (se bild 6.2.2.2). Tornens främsta mål är att skjuta på fienden med projektiler. Det innebär att torn skapar projektil-objekt där tornet anger vilken skada, vilken fiende och var projektilen startar. Efter att projektilen är skapad av tornet hanteras objektet separat. De främsta egenskaperna hos ett torn är attack Hastighet, kostnad och räckvidd.

De torn som finns i spelet och ärver från Tower är:

AirplaneTower – Ett torn med överskriven supermetod move() med extra logik för att röra sig i cirklar runt den ursprungliga positionen. Tornet skjuter StickyProjectile.

CanonTower – Ett torn som skjuter sakta och med låg räckvidd men med PenetratingProjectile. Efter tornet uppgraderats tre gånger ersätts PenetratingProjectile med ExplodingProjectile.

ArrowTower – Skjuter tre stycken ArrowProjectile i snabb följd.

MachineGunTower – Skjuter snabbt, med stor räckvidd och använder BulletProjectile och gör låg skada per projektil.

Bild 6.2.2.2 med towers paketet. Notera att Tower ärver ovanifrån av EntityAttacker. EnemyType beskrivs i början av 6.2

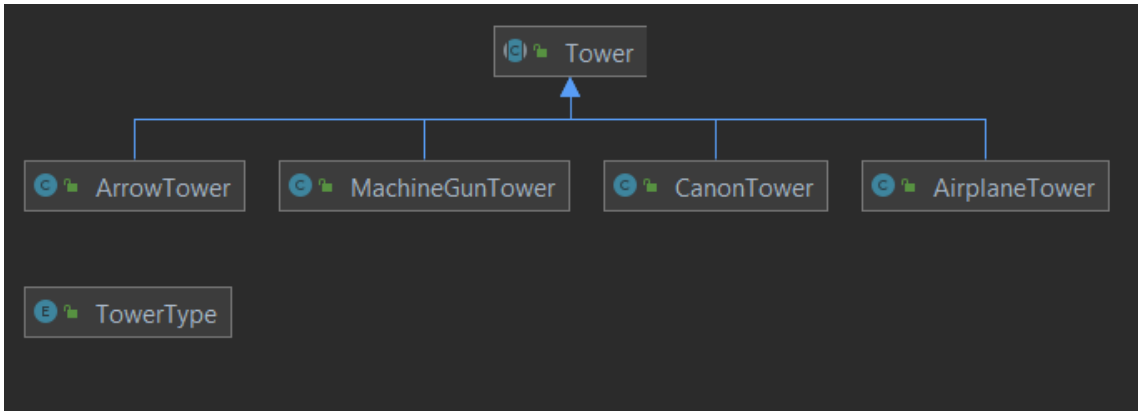

Bild 6.2.2.2

Den abstrakt projektil-klassen Projectile hanterar främst hur projektiler ska röra sig samt när en projektil får attackera en fiende (se bild 6.2.2.3). Projectile skriver över move() metoden ärvd av EntityAttacker (som i sig ärver den från Entity) och lägger till extra logik för att inte stanna eller ändra riktning då den närmar sig sitt mål.

Projektiler kan därför missa den Entity som var målet och eventuellt träffa en annan. Projektiler har dessutom fältet penetrationAmount som representerar hur många Entity objekt den kan penetrera innan ska tas bort.

De klasser som ärver från Projectile och kan användas i spelet är:

StickyProjectile – En målsökande projektil. Projektilen fastnar på sitt targetEntity och landar på marken när det objektet försvinner.
ExplodingProjectile – En långsam projektil som exploderar när den träffar en Entity och skadar de som är inom explosionen.

ArrowProjectile – En projektil med mellan hastighet jämförst med andra projektiler som penetrerar två Entity objekt.
PenetratingProjectile – En långsam projektil som penetrerar 20 Entity objekt.
BulletProjectile – En snabb projektil som inte penetrerar någon Entity.

Bild 6.2.2.3 med projectiles paketet. Notera att Projectile ärver ovanifrån av EntityAttacker. EnemyType beskrivs i början av 6.2

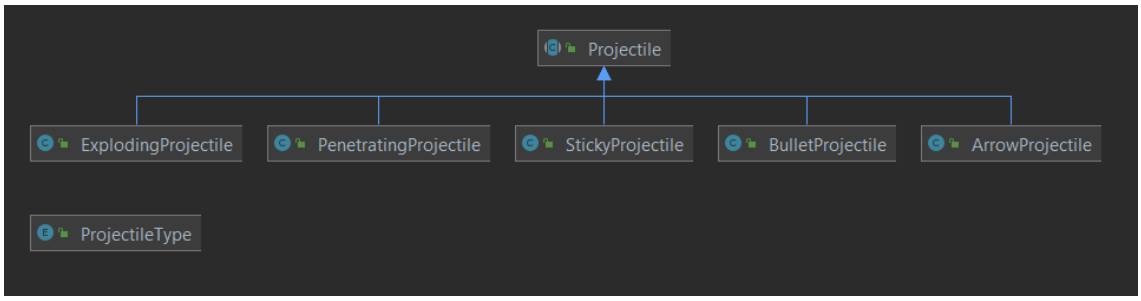

Bild 6.2.2.3

De fienden som skapas ärver från klassen Enemy (se bild 6.2.2.4). Denna klass har den

huvudsakliga specifika funktionaliteten hos fienden. Detta kan vara att fienden rör sig längs med vägen (“move()”), om fienden har gått i mål (“isFinished()”) samt hur den ritas ut (“draw()”)(se bild 6.2.2.4).

För att en fiende ska röra sig längsmed den förutbestämda vägen används fiendens egen metod move(). Denna metod uppdaterar ett fält (“pathProgress”) som anger hur långt fienden har rört sig längsmed vägen. Detta gör att GameHandler konstant vet vilken koordinat fienden ska röra sig mot genom att anropa “getPathProgress()”.

Ett fåtal fienden har egenskapen att de är delbara. Detta innebär att när fiendens liv når noll, skapar fienden ett bestämt antal nya fienden-objekt. För att en fiende ska vara delbar har den fält som anger vilken typ av fiende som ska skapas samt mängd. Delningen sker genom att GameHandler anropar metoden split() efter fiendens liv når noll.

De fienden som finns i spelet är följande:

StandardEnemy – En standardfiende med lågt liv.

SpeedEnemy – En snabb fiende med lågt liv som gör mycket skada på spelaren.

BossEnemy – En långsam fiende med mycket liv och gör mycket skada på spelaren. När fiendens liv når noll skapas fem stycken standarEnemy fiender.

ExplodingEnemy – En fiende som när dess liv når noll skapar fyra SpeedEnemy.

FlyingEnemy – En fiende som ignorerar vägen och rör sig direkt mot slutet på kartan

Bild 6.2.2.4 med enemies paketet. Notera att Enemy ärver ovanifrån av Entity. EnemyType beskrivs i början av 6.2

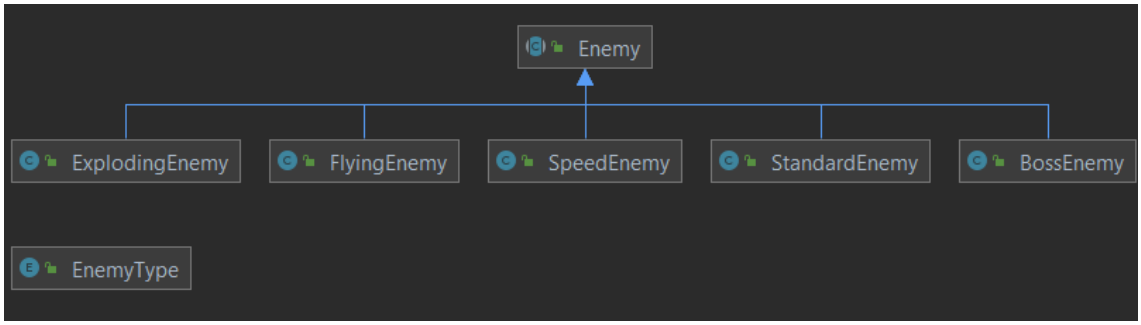

Bild 6.2.2.4

6.2.3 Spellogiken och klassen GameHandler

När användaren har valt karta skapar klassen StartMenu (se 6.2.4) GameHandler. GameHandler tar in GameMap objektet (se 6.2.1) som har laddat spelarens valda karta som parameter till konstruktorn. Spelet uppdateras i takt med en timer genom att timern har konstruerats med ett ActionEvent objekt DoOneStep. DoOneStep finns definerat i GameHandler som en inre klass. GameHandlers tick() metod kallas då varje gång timern aktiveras och leder till att spelet drivs framåt.

GameHandler är klassen som hanterar spelet och den huvudsakliga spellogiken. Detta innebär att det är denna klass som har informationen om spelarens pengar och liv men även torn, projektiler samt fiender.

GameHandlers tick() är den metod som får spelet att fungera. Varje gång metoden tick() aktiveras inspekteras hur mycket liv spelaren har (“inspectHealth()”), torn attackerar och därmed skapar projektiler (“activateTower()”). Projektilerna rör sig

därefter i dess bestämda rikning (“moveProjectiles()”). Likt projektiler skapas fienden samt rör sig (”createEnemies()” och “moveEnemies()”). Därefter uppdateras skärmen genom att notifiera de grafik komponenter som lyssnar (“notifyListernes()”).

Det är GameHandler som lagrar information både om de Entity-objekt (se 6.2.2) som skapas samt om kartan. Detta ger att den funktionalitet som kräver interaktioner mellan Entity-objekt och interaktion med kartan hanteras av denna klass. Detta kan exempelvis vara att torn får de fiender som kan attackeras (“activateTowers()”) eller att fienden får nästa punkt längs med vägen den ska röra sig mot (“moveEnemies()”).

Det är även GameHandler som ansvarar för att nya fienden ska skapas. Detta sker med klassen WaveMaker och dess metod update(). Denna metod skapar fienden i vågor som består av ett flertal etapper. Olika fiender skapas i olika vågor, exempelvis så skapas endast StandardEnemy (bastypen av fiende) de första 5 vågorna.

6.2.4 Start av spel, användargränssnitt och grafik

När användaren väljer att starta ett spel är det klassen GameStarter som startas. GameStarter initierar en logger som skriver ner de kritiska händelserna exempelvis laddning av karta men även om något fel uppstår. GameStarter skapar sedan en startmeny med klassen StartMenu. Startmenu objektet skapar startmenyn med metoden createStartMenu(). Startmenyn består utav knappar med standardkartorna utritade på sig. Användaren kan välja vilken karta som spelat ska startas på genom att klicka på den knapp där önskad karta finns utritad. Då användaren tryckt på en av dessa knappar laddas GameMap med den karta som knappen representerar. GameMap skickas därefter som parameter till ett nytt GameHandler-objekt. Startmenu initierar ett nytt GameViewer objekt med GameHandler-objektet som parameter till konstruktorn. Efter detta startas spelet.

Grunden för grafiken bygger på användargränssnittshanteraren “Java Swing”. Det är Swing som hanterar det fönster som visar spelet. För att strukturera gränssnittet i olika delar används paneler med Layout-hanteraren GridLayout. GameViewer använder sig av två huvudpaneler, varav den första heter gamePanel och den andra panelen heter menuPanel.

GamePanel innehåller ett objekt av klassen GameComponent som ritar ut spelet med torn, tiles, projektil och fienders egna draw() metoder.

Den andra panelen menuPanel innehåller information om spelets tillstånd samt knappar. MenuPanel är uppdelat i fyra stycken paneler varav dessa innehåller olika delar av menyn. De paneler som finns i menuPanel följer nedan.

TowerDescriptionPanel har en textruta med information om det torn spelaren har klickat på, exempelvis om spelaren klickar på ett torn. Där visas information om tornets attack-skada, räckvidd, start- samt uppgraderingskostnad. TowerDescriptionPanel är även i en JScrollPane, vilket gör att om mer information om tornet ges än vad som ryms i panelen går panelen att skrolla.

I textPanel finns en komponent vid namn MenuComponent som ritar ut spelarens liv, antal pengar spelaren har samt vilken nivå spelet befinner sig på.

InteractivePanel innehåller de knappar som spelaren kan trycka på för att köpa ett torn. Knapparna i InteractivePanel har en ButtonGroup där alla knappar relaterad till

att köpa torn finns tillagda. Gruppen gör att när en knapp inom gruppen trycks ned kommer resterande knappar att avtryckas. InteractivePanel är dessutom i en JScrollPane, vilket innebär att ifall fler typer av torn läggs till i spelet än vad som ryms i panelen går panelen att skrolla. När en knapp trycks används ButtonEvent klassen.

ButtonEvent konstrueras med två enum typer TowerType och ButtonType. ButtonType anger vilken typ knappen är, i detta fall TOWER_BUTTON. TowerType representerar vilket torn som knappen ska skapa när den trycks.

TowerUpgradePanel består utav en knapp användaren kan trycka på för att uppgradera det torn som användaren valt. Denna knapp fungerar på likande sätt som i InteractivePanel. Samma konstruktor i ButtonEvent tar in Enum-typen UPGRADE som parameter för att avgöra vilken knapp som har tryckts.

Den sista panelen heter PauseAndQuitPanel. Denna panel består utav en växlingsknapp som startar respektive pausar spelet. Panelen innehåller även en knapp som avslutar spelet. Dessa knappas använder även ButtonEvent klassen men ger Enum parameterena PAUSE respektive QUIT.

Ett torn placeras med ett mustryck genom att en muslyssnare “MouseEvent” finns för gamePanel. När spelaren klickar på gamePanel (spelplanen) skapas ett MouseEvent. MouseEvent objektet innehåller information om var på skärmen spelaren har tryckt. Med koordinaterna placeras ett torn om ett torn är valt från menyn, ifall ett torn redan finns på dem koordinaterna visas istället dess information i towerDescriptionPanel.

MenuComponent och GameComponent använder sig av gränssnittet GameListener (se bild 6.2.4.1). I GameListener finns de metoder som är nödvändiga för att veta när klassen GameHandler meddelar en grafisk ändring och därmed att en ny utritning av spelet ska ske. Notera även att komponent objekten måste finnas i GameHandlers lyssnar lista för att uppdateras då GameHandler meddelar ändring.

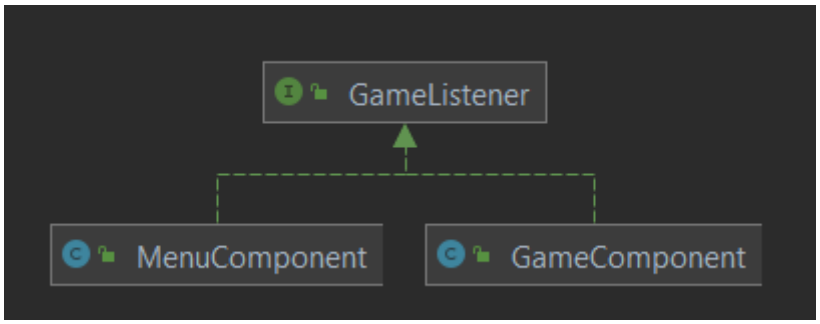

Bild 6.2.4.1

7. Användarmanual

Vid spelets start syns bild 7.1 där spelaren väljer vilken karta som ska spelas på.

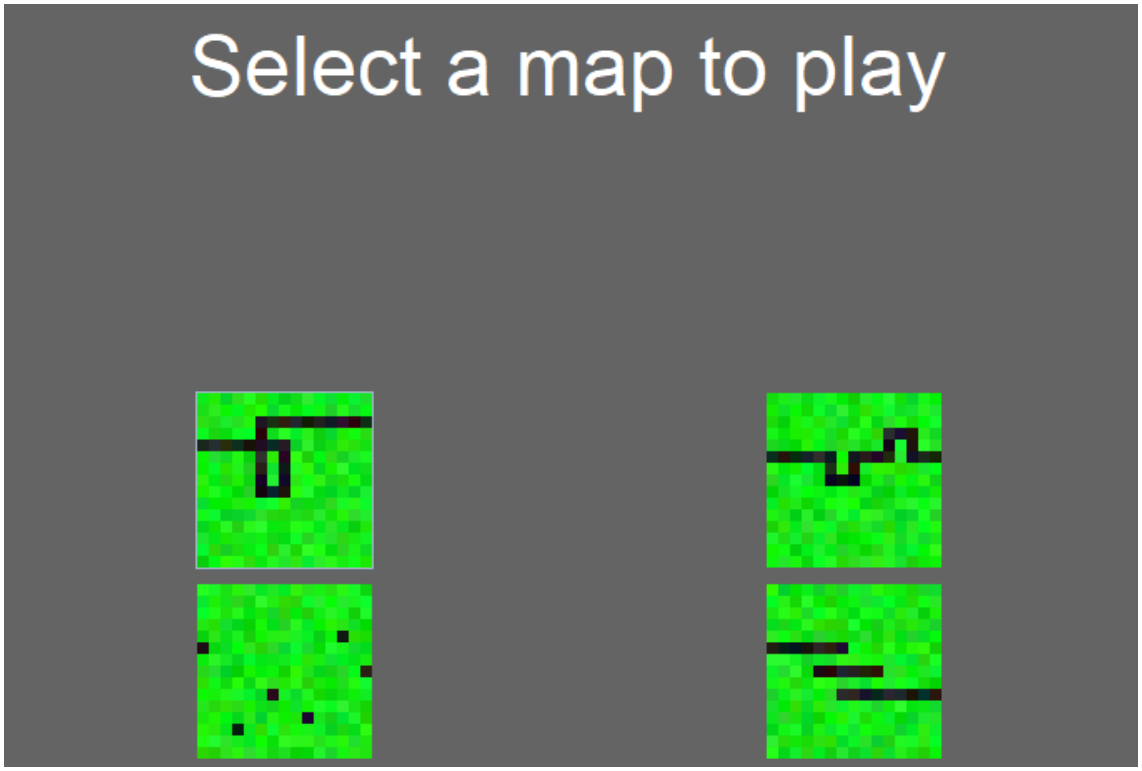

Bild 7.1

När spelaren klickar på en karta så kommer kartan att laddas och spelet startas på vald karta (se bild 7.2).

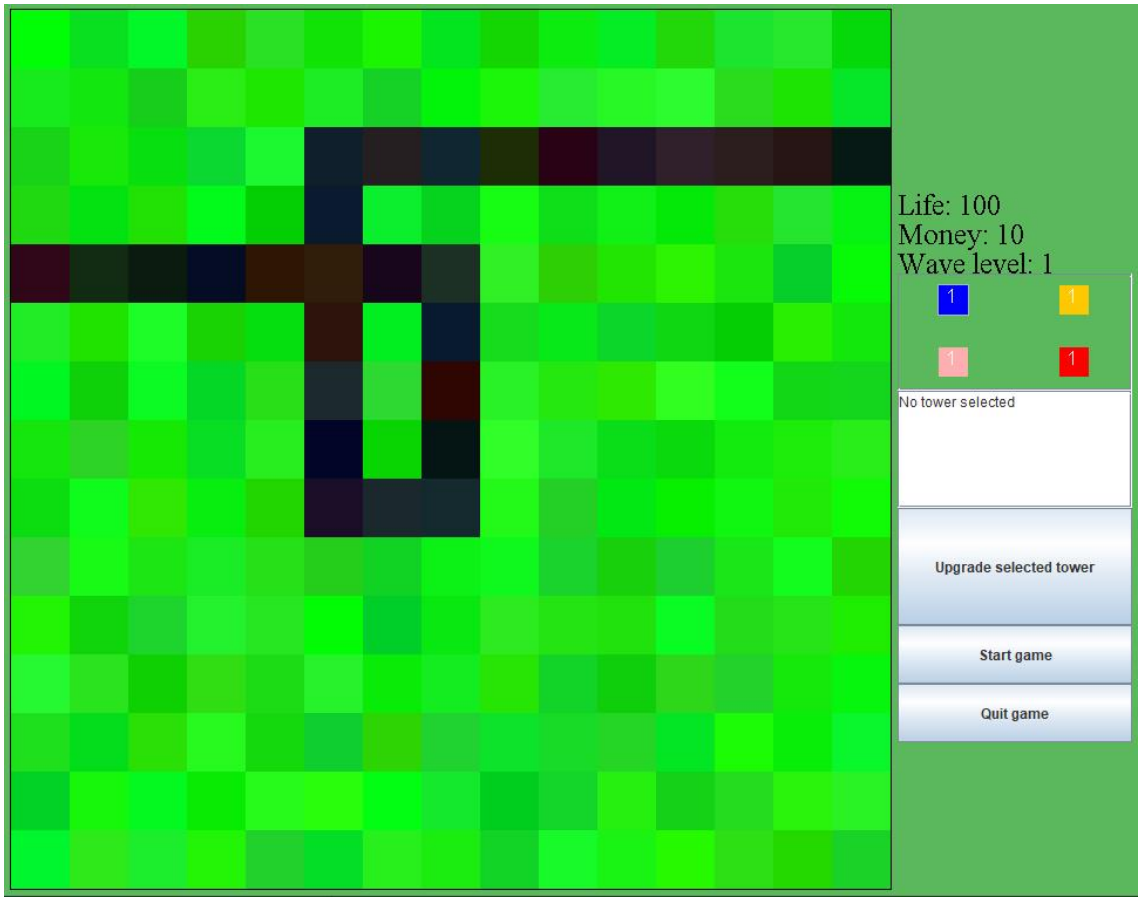

Bild 7.2

För att starta spelet trycker spelaren på knappen “Start game”.

Målet är att försvara mot de fiender som rör sig från början till slutet av den svarta

vägen. Om en fiende når slutet av vägen tar spelaren skada, när spelarens liv är noll är spelet över.

För att försvara mot fienderna används de torn som finns tillgängliga i menyn till höger representerade av färgade kvadrater. För att placera ett torn klickar spelaren på ett av de tillgängliga tornen och klickar sedan på ett gräsblock (se 6.2.1). Tornen har däremot begränsad räckvidd och bör därför placeras nära vägen. I spelet kostar torn pengar vilket kan erhållas genom att förstöra fienden, eller från det startkapital som ges i början av varje spel.

För att köpa torn används menyn (se bild 7.3) där visas kostnad, räckvidd, skada och hur många spel-tick det tar för ett torn att skjuta så kallat *"attack speed"*

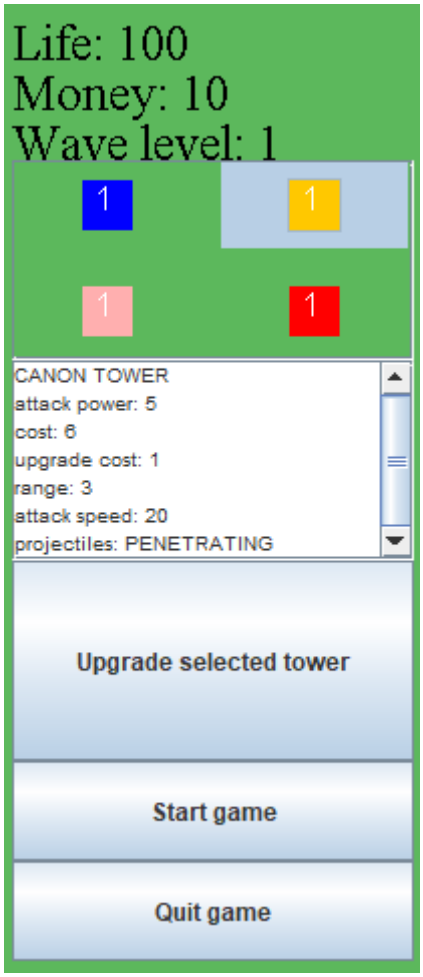

Bild 7.3

Det finns även möjlighet att uppgradera de torn spelaren har placerat. Detta görs genom att klicka på ett torn som finns utplacerat och tycka på *"upgrade selected tower"*. Att uppgradera torn kostar pengar, kostanden visas i menyn till höger inom textrutan som beskriver det valda tornet. Efter uppgradering ökar nivån för tornet som också ritas ut på tornet, dessutom ökar kostnaden för ytterligare uppgradering. När tornet är valt visas även dess räckvidd (se bild 7.4).

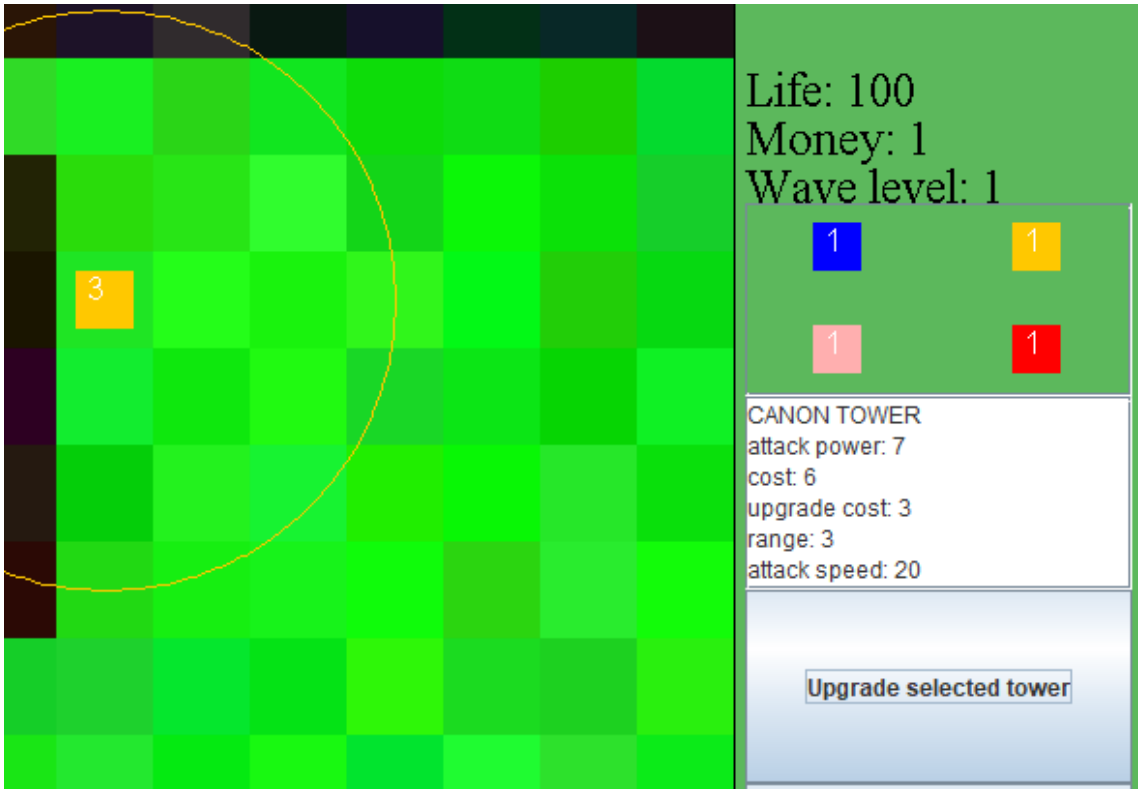

Bild 7.4

Nedan visas ett exempel på hur spelplanen kan se ut under en spelomgång (se bild 7.5).

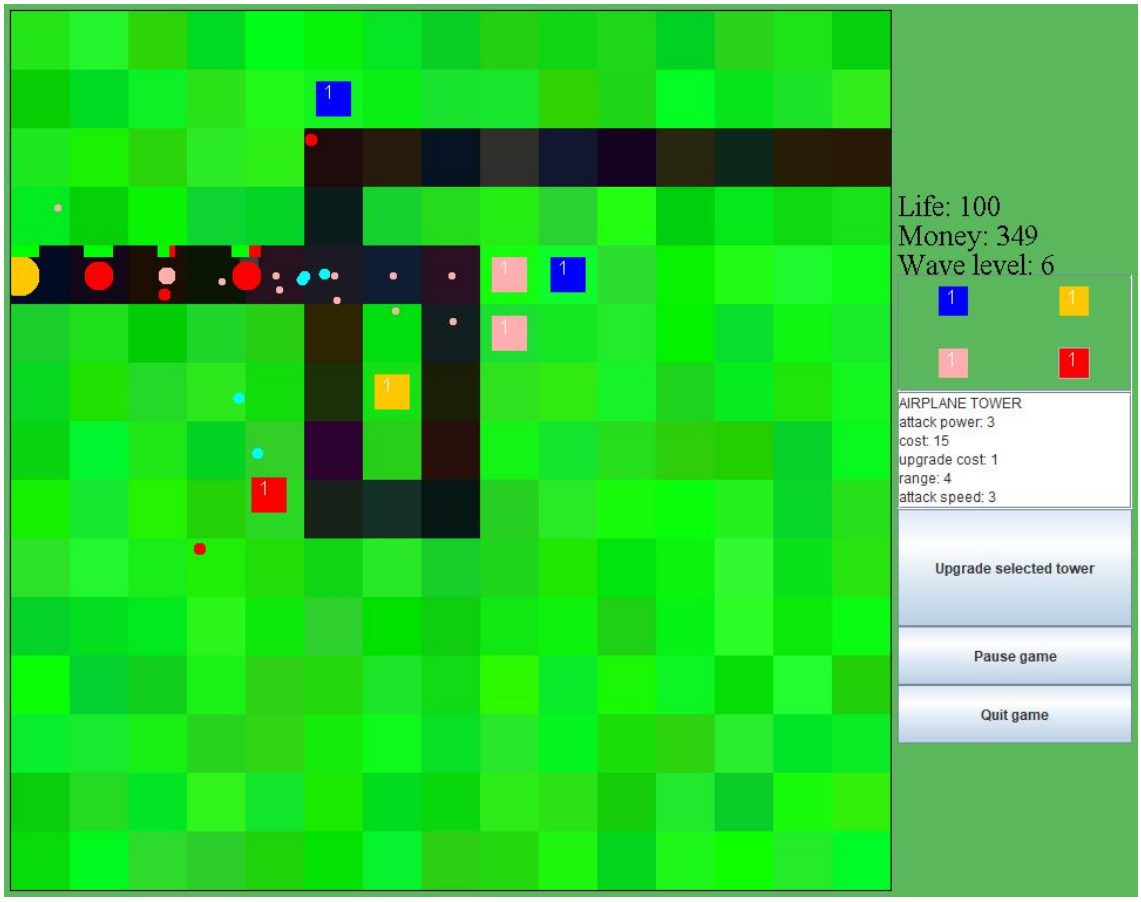

Bild 7.5