Tower Defence

Erfarenhetsrapport

Christoffer Gustafsson
Henrik Holmberg
Magnus Holmbom
Filip Jaredson
Kevin Lindström
Philip Pettersson

Detta projekt har varit en lärorik upplevelse för hela gruppen på många olika sätt. Tillsammans har vi utvecklats som både mjukvaruutvecklare och medlemmar i ett grupparbete, och på vägen fått vara med om många nya erfarenheter och situationer vi inte tidigare mött i vår utbildning, vilket vi vill beskriva nedan.

Till att börja med har vi inte gjort mjukvara i sådan här utsträckning förr. Spelet hade ett skelett som bestod i många delar, bl.a. tillståndshantering, torn, monster, och en mängd underklasser till dessa som utvidgar dess funktionalitet. Detta var vår första utmaning, då vi efter ett tag märkte att alla spelfiler behövde ha tillgång till vårt speltillstånd, eller Game State. Frasen “Är det någon som sitter i Game State? Kan jag pusha?” har ekat i labbsalarna i flera veckor sedan projektets begynnelse, och hade vi fått göra om projektet från början hade vi försökt göra spelet mer modulärt från första början så vi sluppit onödiga konflikter i vår Git-repository.

Det storskaliga projektet har också gett ett stort fokus på samarbete med andra medlemmar i projektet. Många delar av spelet beror på vad andra sysslar med, och utöver att man har gemensamma filer så behöver man förstå vad det är andra i gruppen har jobbat med. Att sätta sig in i funktionaliteten på grundlig nivå hos varje klass hade varit svårt, så istället tog vi hjälp av varandra och förklarade hur man använder den, samt diskuterade vad som behövs när andra behöver tillgång till koden. Vi har därför tyckt det varit bäst när vi programmerat tillsammans, då det blir lätt att ta hjälp av varandra och diskutera med andra i gruppen när vi fastnat eller inte förstått deras källkod.

Nedan följer några individuella erfarenheter från medlemmarna i gruppen:

Filip:

Under kursens projekt har ju stort fokus varit på hur man implementerar och hanterar objektorienterad programmering i C++. Innan kursen har jag kodat ganska mycket objektorienterat i Scheme när jag undervisat i kursen som tidigare gick för Y-programmet, men arbetssättet och tankegången när man jobbar med OOP i C++ skiljer sig en hel del från det. Bruket av pekare och skillnaden i hur information lagras har varit en stor hjälp för projektet, då det har förenklat överföring av information. Stödet som finns tillgängligt på forum så som StackOverflow för C++ är mycket bättre och bredare än det till Scheme, vilket har möjliggjort mycket djupare inlärning och förståelse av hur man kan implementera saker på olika sätt. Mängden olika svar på liknande frågor har även gjort det möjligt att välja en lösningsgång man själv tycker möter ens behov bäst, och det har möjliggjort utvidgning av de lösnings gångarna till den lösning man själv är ute efter.

Magnus:
Under OOA/OOD borde vi ha lagt mer tid på att ta fram ett individuellt testsystem (överhuvudtaget funderat mer på hur vi skulle testa). Vi har använt testsystemet CATCH, men det har varit problematiskt ty vissa klasser har berott väldigt mycket på varandra. När man gjort ett TestCase i CATCH har det varit så att vissa funktioner saknats i en annan klass man

behöver använda, andra funktioner ändras, det finns bara en "pure virtual"-version av klassen osv. Detta har resulterat i att testfilen man gjort, som en gång gav tillfredsställande resultat, vid ett senare tillfälle gett kompileringsfel.

Det slutade med att alla testade sina klasser i "Game_State.cpp", det är filen som ser till att allt i spelet uppdateras, renderas och håller koll på många av objekten som skapas i spelet. För att undvika merge-konflikter har inte fler än en gruppmedlem redigerat i filen åt gången, vilket lett till ineffektiv testning.

Christoffer:
SFML har varit vårt val av multimedia-API (applikationsprogrammeringsgränssnitt). Det har varit lätt att hitta hur man implementerar de inbyggda klassfunktionerna på deras hemsida. När vi sitter lokalt så har jag sällan erfarit lagg eller några buggar. Att vi använt oss av API som rekommenderats för kursen gjorde det möjligt att fråga kursassistenten om man inte förstod hur något skulle implementeras.

Kevin:
När vi gjorde vår OOD och OOA satt samtliga gruppmedlemmar ofta tillsammans för att brainstorma. Det var väldigt tidsineffektivt och tog mycket längre tid än jag trott. Hela gruppen lade totalt ungefär 240 timmar på bara denna del. I framtiden kommer jag nog föreslå en annan metod, t.ex. att brainstorma i mindre grupper först.

Henrik:
Projektet har använt sig av "Christoffers spelskelett med tillståndsmaskin" som kommer med en färdig Makefile. Till skillnad mot tidigare då man manuellt var tvungen att lägga in alla filer som skall användas i Makefile, möjliggjorde denna att dynamiskt ta med alla filer automatiskt, vilket förenklar arbetet. Detta gav även den typiska uppsättningen med header-filer i mappen Include, source-filer i mappen Src och alla kompilerade filer i Bin.
Efter lite modifieringar klarade vår Makefile även att ta hand om test-fall där test-filerna sållas bort vid vanlig make

Philip:
“Att spawn:a motståndare”. Innan projektet har jag inte reflekterat mycket över hur motståndare skapas och sköts i spel. Till vårt Tower Defense spel skapas motståndare med en textsekvens. En spawn-pump tolkar sekvensen och spottar motståndare till en container. Huvudprogrammet kan sedan gå genom containern så att motståndarna stegas fram, kollisionshanteras och ritas ut. I vår implementation att skapa motståndare åsido, finns det många spännande sätt att skapa motståndare. Istället hade antalet motståndare och deras tålighet kunnat slumpats ut eller baserats på spelarens möjlighet att stoppa motståndarna. Projektet har givit mig nya ögon på att skapa motståndare i spel.