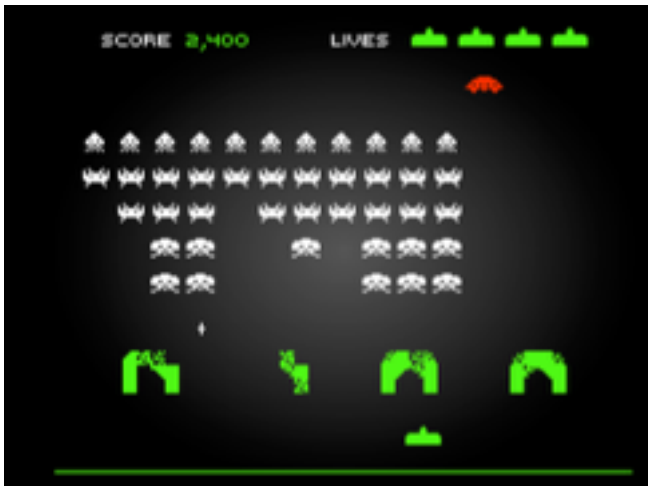
Project Gevorderd Programmeren 2015 - 2016

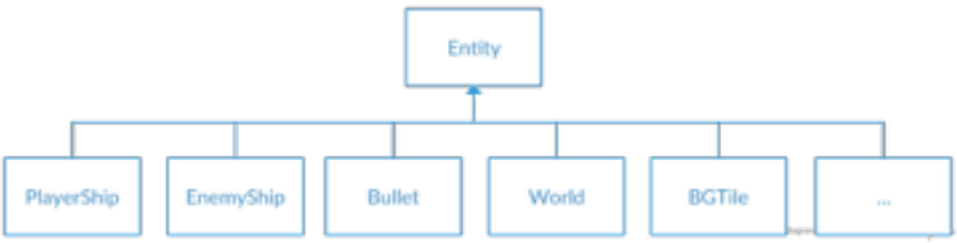


Inleiding

Het doel van de project is het ontwerpen en implementeren van een interactief spel geïnspireerd op Space Invaders. De speler bestuurt een verdedigingskanon die de opdracht heeft om de inkomende aliens te vernietigen. De alien-ships dalen neer in zig-zag bewegingen en trachten tevens de speler neer te halen. De speler kan onderaan het scherm horizontaal bewegen en zich eventueel verschuilen achter “schilden” die ook kunnen worden kapotgeschoten.
De speler verliest het spel wanneer de aliens de schilden (of de aarde) bereikt hebben of wanneer de speler wordt neergeschoten. De speler wint de huidige level wanneer alle aliens zijn neergehaald. Daarna begint de volgende level (met snellere aliens/kanonnen). Af en toe krijgt de speler de mogelijkheid om power-ups te pakken die hem een snellere kanon bezorgen of de vijanden even doet stoppen.
Kijk eens op http://www.freeinvaders.org voor wat meer inspiratie. Vanzelfsprekend mag je de op hierboven vernoemde spelregels met de nodige flexibiliteit interpreteren en zelf in-/aanvullen.

Implementatie

De nadruk van dit project ligt op een elegant ontwerp van de game entities en het correcte gebruik van de vereiste design patterns. Ontwerp een klasse structuur voor de spel entiteiten (World, Entity, Playership, Enemyship, Bullet(s), BGTile, …) die je in staat stelt om dat te doen. Houd hierbij rekening met de uitbreidbaarheid van uw structuur. Bijvoorbeeld het mag niet al te moeilijk zijn om een nieuw type alien/spaceship/laserkanon/bullet of power-up te ontwikkelen of om een multi-player mogelijkheden toe te voegen.



Gebruik de nodige features in C++ om je hierbij te helpen. Zaken die aan bod **moeten komen**, zijn:

- **Afgeleide klassen** en **polymorfisme**

- Een **Model-View-Controller (MVC**) ontwerp voor de interactie tussen de game-logic, grafische weergave in SFML en de interactieve speler. Gebruik een **Observer pattern** voor het updaten van de **View(s)** bij veranderingen in het **Model**. **Dit zou je de mogelijkheid moeten geven om de visualisatie volledig te kunnen scheiden van de logica van het spel.**

- Implementeer een eenvoudige **Stopwatch** klasse (die de tijd tussen twee "ticks" bijhoudt omdat niet alle computers dezelfde snelheid hebben en toch moeten je entities op alle computers even snel bewegen) en **RandomGenerator** klasse volgens het **Singleton pattern**.

- Gebruik de **C++ standard library** waar nodig en/of nuttig.

- Gebruik **libSFML** om de grafische implementatie & input/output te voorzien (dezelfde versie als op de lab computers). Meer informatie over SFML vind je op http://www.sfml-dev.org en in sessie 3 van de practica.

- Gebruik **namespaces** om het modulair design duidelijk aan te geven. Wrap alle jouw klassen in een gezamenlijke **si** namespace (de "space invaders" namespace) en de verschillende componenten (model, view, controller) in geneste namespaces (bijvoorbeeld, **si::model**).

- Gebruik **exception handling** voor het opvangen en afhandelen van eventuele fouten (i.e., inlezen van image files, initialisatie van grafische omgeving, lezen van een level file, …)

- Voorzie **meerdere levels** die kunnen gespeeld worden. Deze levels moeten worden uitgelezen uit een file (bijvoorbeeld XML, JSON, …) aan de hand van een externe library (bijvoorbeeld met de BOOST library of indien een andere library wordt gebruikt en deze *niet* aanwezig is op de referentiecomputers, **moet deze library aan het project worden toegevoegd**.)

- Focus niet te veel op ingewikkelde AI van de enemies en hou de collision handling in de wereld eenvoudig (je kan veronderstellen dat de objecten cirkelvormig zijn).

- Doe vooraf wat leeswerk in verband met game design en de verplichte design patterns:
  - http://www.gamasutra.com/view/feature/2280/the_guerrilla_guide_to_game_code.php
  - http://www.mine-control.com/zack/patterns/gamepatterns.html
  - http://content.gpwiki.org/index.php/Observer_Pattern
  - https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller
  - https://en.wikipedia.org/wiki/Observer_pattern
  - https://en.wikipedia.org/wiki/Singleton_pattern
  - …

Rapport

Beschrijf uw design en de keuzes die je hebt gemaakt in een schriftelijk **verplicht** rapport van **2 A4 pagina's**. Hieruit moet duidelijk zijn dat je weloverwogen keuzes hebt gemaakt om jouw design samen te stellen en dat je deze keuzes kan verantwoorden. Voorzie indien nodig ook UML diagrammen **in bijlage** om je design toe te lichten. Andere toevoegingen in het rapport kunnen eventueel geïmplementeerde extensies zijn, spelregels, spelbesturing uitleg, …

Praktische afspraken

- Een compilerende en werkende versie van het spel met de eerder vernoemde features, ontworpen volgens de principes van goede software design in C++ is genoeg om een voldoende cijfer te behalen. Concentreer je hierbij op de volgende zaken:
  - **Logisch en overzichtelijk ontwerp en implementatie van de klassen** en hun interacties. Volg de principes die je hebt geleerd in de les.

  - **Duidelijke documentatie** van de code. Zowel de API van jouw klassen als de minder voor de hand liggende stukken code dienen telkens duidelijk goed gedocumenteerd te worden.

  - Lay-out van het project: hou je aan een **overzichtelijke en logische directory structuur** voor de code, build, image files, … Gooi niet alles in één map, maar maak er ook geen doolhof van mappen van.

  - **Werk incrementeel**. Schrijf geen honderden lijnen code met de hoop dat het uiteindelijk wel zal compileren en werken. **Implementeer eerste de minimale vereisten en denk daarna aan eventuele uitbreidingen! Je mag creatief zijn, maar focus erop dat het spel werkt.** Je mag zelf kiezen wel uitbreidingen je (niet verplicht) nog extra implementeert:
    - Verschillende types enemy ships
    - Verschillende kanonnen waarmee er kan geschoten worden
    - Highscores
    - Explosieanimaties
    - … (online games kunnen inspiratie brengen)

    - Gebruik **CMAKE** als built-systeem. Voorzie daarbij ook een **run.sh (bash/shell) script** dat bij uitvoering de CMAKE commando's uitvoert en het spel opstart.

- **LET OP:** een **niet-compileren/werkend** project (bijvoorbeeld, compile errors of een segmentation fault bij opstarten) betekent automatisch "niet geslaagd voor dit onderdeel". **Als referentieplatform worden de computers in het lab G026 gebruikt. Het project moet hierop compileren en werken.** Test daarom je spel op voorhand (**minstens** 1 week voor de deadline!) op deze computers.

- Het project dient **zelfstandig** gemaakt en ingeleverd te worden. Je mag natuurlijk naar hartenlust jouw design en mogelijke problemen en oplossingen **overleggen** met anderen.

- Succes! Indien je problemen hebt, aarzel dan niet om me (**glenn.daneels@uantwerpen.be**) te contacteren of langs te komen in mijn bureau G212.

- De deadline van het project (inc. rapport) ligt vast op **3 dagen voor het theoretisch examen Gevorderd Programmeren**. Dit zal vermoedelijk halverwege januari 2016 zijn.

- Het project moet zowel **via Blackboard als via email** (**glenn.daneels@uantwerpen.be**) worden ingediend. De naam van jouw project is naamstudent_rolnummer.tar.gz (of .zip).

Succes!