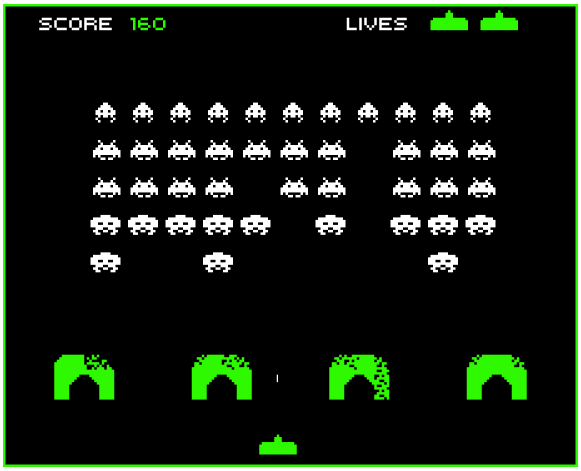
Space Invaders

project C++ 2013 - 2014



Inleiding

Het doel van dit 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 hierboven vernoemde spelregels met de nodige flexibiliteit interpreteren en zelf in/aanvullen.

Details i.v.m. implementatie

De nadruk in 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 die je in staat stelt om dat te doen. Hou hierbij rekening met de uitbreidbaarheid van uw structuur. Bv.: het mag niet al te moeilijk zijn om een nieuw type alien/spaceship/laserkanon/bullet of power-up te ontwikkelen of om 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** 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 de **Model**. Dit zou je de mogelijkheid moeten geven om de visualisatie volledig te kunnen scheiden van de logica van het spel.
- 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 info over hoe & wat met SFML vind je op: http://www.sfml-dev.org/

- Gebruik **Namespaces** om het modulair design duidelijk aan te geven
- Gebruik **Exception handling** voor het opvangen en afhandelen van eventuele fouten (i.e.: bij inlezen van image files, initialisatie van de grafische omgeving, lezen van een level file ...)
- Gebruik van een **abstract factory design pattern** voor het aanmaken van alle game-entities.
- Ik moedig je ten zeerste aan om wat leeswerk vooraf te doen in verband met design van games, bijvoorbeeld (telkens korte artikels):
    - 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
    - http://en.wikipedia.org/wiki/Model-view-controller
    - http://en.wikipedia.org/wiki/Observer_pattern

Report

Beschrijf uw design en de keuzes die je hierbij moest maken in een schriftelijk (**pdf**) report van **2 A4 pagina's**. Hieruit moet duidelijk zijn dat je weloverwogen keuzes hebt gemaakt om jouw design samen te stellen. Voorzie indien nodig ook UML diagrammen **in bijlage** om je design toe te lichten.

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 goed gedocumenteerd te worden.
    - Lay-out van het project: hou je aan een **propere directory structuur** voor de code, build en eventuele support files. Gooi niet alle files in één map; maak er anderzijds 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 eerst de minimale vereisten en denk daarna aan eventuele uitbreidingen**.
    - Gebruik **CMake** als build-systeem.
- Let op: een **niet-compilerend/werkend project** (bv. compiler errors of een "segmentation fault" bij opstarten) betekent automatisch "niet geslaagd voor dit onderdeel". Als **referentieplatform** worden de computers in het lab G026 gebruikt. **Uw code moet daar compileren & werken.**
- Het project dient **zelfstandig** gemaakt en ingeleverd te worden. Je mag natuurlijk naar hartelust jouw design en mogelijke problemen en oplossingen **overleggen** met anderen.
- Succes! In geval van vragen of opmerkingen over dit project kan je mij telkens bereiken via przemyslaw.klosiewicz@uantwerpen.be of je kan mij vinden in bureau **G207**.
- De **deadline** van het project (incl. report) ligt vast op **3 dagen voor het theoretisch examen** Gevorderd Programmeren. Dit zal vermoedelijk halverwege Januari 2014 zijn.
- **Lever het project (incl. report) zowel via BlackBoard als per mail in**: (przemyslaw.klosiewicz@uantwerpen.be)