Université de Montréal

Dévelopment de jeux vidéo en Scheme

par
David St-Hilaire

Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences

Mémoire présenté à la Faculté des études supérieures
en vue de l'obtention du grade de Maître ès sciences (M.Sc.)
en informatique

Décembre, 2009

Université de Montréal
Faculté des études supérieures

Ce mémoire intitulé:

Dévelopment de jeux vidéo en Scheme

présenté par:

David St-Hilaire

a été évalué par un jury composé des personnes suivantes:

Mostapha Aboulhamid
président-rapporteur

Marc Feeley
directeur de recherche

Yann-Gaël Guéhéneuc
membre du jury

Mémoire accepté le

RÉSUMÉ

Mots clés: Language de programmation fonctionnels, Scheme, jeux vidéo, programmation orientée objet.

ABSTRACT

Keywords: Functional programming languages, Scheme, video games, object oriented programming.

TABLE DES MATIÈRES

LISTE DES FIGURES

REMERCIEMENTS

blablabla

CHAPITRE 1

INTRODUCTION

1.1 Contexte

1.1.1 jeu video -¿ $$

1.1.2 prog de jeu complexe : bcp années/hommes de travail

1.2 Motivation

1.2.1 Comment faciliter le dév ?

1.2.2 Prog haut niveau vs bas niveau

1.2.3 Scheme

1.3 Problématique

Ce mémoire de maîtrise vise à répondre à la problématique suivante :

Est-ce possible ou envisageable de concevoir et développer des jeux vidéo en Scheme ? Quels en sont les avantages et les inconvénients ?

1.4 Méthodologie

1.4.1 Dev 1er jeu simple pour determiner les besoins pour Scheme

1.4.2 Augmenter Scheme pour repondre a ces besoins

1.4.3 Ecrire un nouveau jeu utilisant les techniques developpees pour le 1er jeu

Afin de pouvoir répondre à ces questions, 2 jeux vidéo ont été developpés en utilisant le langage de programmation Scheme. Le premier jeux a servi de platforme d'exploration permettant d'élaborer une méthodologie qui semble efficace pour le développement de jeux. Afin d'obtenir une telle méthodologie, plusieurs itérations de développement ont été effectuées, chacune permettant d'explorer de nouveaux aspects sur la manière de résoudre les problématiques associées à la création de jeux, comme par exemple comment arriver à synchroniser des entités dans le jeux ou comment arriver à décrire efficacement un système de détection et de résolution de collisions.

Suite à l'écriture de ce premier jeu, un autre jeu plus complexe que le premier a été écrit afin de consolider les méthodologies précédemment utilisées.

CHAPITRE 2

DÉVELOPPEMENT DE JEUX VIDÉO

2.1 Contexte (intro ?)

2.1.1 Grosseur du marche des JV

2.1.2 Diversite des jeux (genres, plates-formes)

2.1.3 Charactéristiques de jeux modernes

Les jeux vidéo font parti d'un domaine de l'informatique en pleine effervescence grâce à une demande constante de nouveaux produits. Ces produits possèdent plusieurs caractéristiques de qualité auxquelles les consommateurs s'attendent à obtenir en effectuant l'acquisition d'un nouveau titre. Ces attentes du consommateur peuvent se traduire par les besoins suivant :

mettre ici une liste des attentes pour un jeu moderne

2.2 Historique

2.2.1 préhistoire 1948-1970

2.2.1.1 CRT games

2.2.1.2 Mainframe games

2.2.2 Système arcade 1970-1985

2.2.2.1 Pong

2.2.2.2 Space Invaders

2.2.2.3 Pac-Man

2.2.3 Premières consoles 1972-1984

2.2.3.1 Magnavox Odyssey

2.2.3.2 Atari XX00 et ColecoVision

2.2.4 Ordinateurs personnels 1977-...

2.2.4.1 Évolution parallèle constante

2.2.4.2 Toujours été jusqu'à la venu des console modernes la plateforme offrant les meilleurs performances.

2.2.5 Consoles portables 1980-...

2.2.5.1 GameBoy etc...

2.2.6 Consoles intermédiaires 1984-2006

2.2.6.1 NES, SNES, N64, GameCube

CHAPITRE 3

LE LANGAGE SCHEME

Référence temporaire pour pas faire planter le makefile [1]

CHAPITRE 4

PROGRAMMATION ORIENTÉE OBJET

CHAPITRE 5

SYSTÈME DE COROUTINES

5.1 Motivation

5.1.1 Contrôle fin du comportement de threads

5.1.2 Parler de l'intérêt de Termite

5.1.3 Motivation de l'utilisation de coroutines (contrôle exact sur le flot de contrôle == système toujours dans un état consistant).

5.2 Description du langage

5.2.1 Création de coroutines

5.2.2 Manipulation du flot de contrôle

5.2.2.1 yield

5.2.2.2 super-yield

5.2.2.3 terminate-corout, kill-all !, super-kill-all !

5.2.2.4 sleep-for

5.2.2.5 continue-with

5.2.2.6 spawn-brother, spawn-brother-thunk

5.2.2.7 Composition of coroutines

5.2.3 Système de communication inter coroutines

5.2.3.1 !

5.2.3.2 ?

5.2.3.3 ? ?

CHAPITRE 6

ÉVALUATION ET EXPÉRIENCES

6.1 Développement de « Space Invaders »

6.1.1 Objectifs

6.1.1.1 Expérimentation avec un jeu très simple

6.1.1.2 Trouver les problèmes fondamentaux pour le développement de jeux

6.1.1.3 Tenter de les résoudre

6.1.2 Version initiale

6.1.2.1 Premier jet dans le but de trouver des problèmes potitiels

6.1.2.2 Comment faire des animations ? =¿ CPS

6.1.2.3 Comment concevoir une partie a 2 joueurs ? =¿ coroutines

6.1.2.4 Difficulté à décrire la résolution de collision de manière efficace

6.1.2.5 Est-il possible d'écrire le comportement d'une entité de manière indépendante, i.e. que le code soit centralisé dans une même fonction ?

6.1.3 Version orientée objet

6.1.3.1 Motivation : Utilisation de fonctions génériques

6.1.3.2 Code Highlight : Résolution de collisions

6.1.4 Version avec système de co-routine

6.1.4.1 Motivation : Intégrer les coroutines a chaque objet de manière à ce que chaque instance soit une entité à part entière qui doit

CHAPITRE 7

TRAVAUX RELIÉS

7.1 Comparaison de langages

7.1.1 Lua

7.1.1.1 Differences entre lua et Scheme

- Lua est de petite taille en mem
- ..

7.1.2 C++

7.1.2.1 Differences entre C++

- Gestion memoire manuelle
- méthode surdéfinies vs fonctions génériques

7.2 Jeux en Lisp

7.2.1 QuantZ

7.2.1.1 A voir avec Robert

7.2.1.2 FRP

7.2.1.3 Techniques anti-gc

7.2.1.4 Delegation de fermetures

7.2.2 Naughty Dogz

7.2.2.1 GOAL

- http ://en.wikipedia.org/wiki/Game_Oriented_Assembly_Lisp
- http ://grammerjack.spaces.live.com/blog/cns !F2629C772A178A7C !135.entry

CHAPITRE 8

CONCLUSION

L'expérience d'écriture de ces jeux aura permis de faire le point sur les avantages et les inconvénients de l'utilisation d'un langage tel que Scheme pour le développement de jeu vidéo.

+ puissance d'expression / d'abstraction

+ langage dynamique (développement en-direct, malléabilités)

+ création de langages spécifiques au domaine

- Garbage Collection et sur-allocation

- Profilage plus difficile avec des LSD (pour Gambit-C et statprof)

- Balance entre abstraction et efficacité

BIBLIOGRAPHIE

[1] Harold Abelson, Gerald Jay Sussman, and Julie Sussman. *Structure and interpretation of computer programs*. MIT Press, Cambridge, Mass., 1996.