

Université de Montréal

## Développement 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

© David St-Hilaire, 2009.

Université de Montréal  
Faculté des études supérieures

Ce mémoire intitulé:

**Développement 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

<b>RÉSUMÉ</b>	<b>iii</b>
<b>ABSTRACT</b>	<b>iv</b>
<b>TABLE DES MATIÈRES</b>	<b>v</b>
<b>LISTE DES FIGURES</b>	<b>xi</b>
<b>REMERCIEMENTS</b>	<b>xii</b>
<b>CHAPITRE 1 : INTRODUCTION</b>	<b>1</b>
1.1 Contexte	1
1.1.1 jeu video -; \$\$	1
1.1.2 prog de jeu complexe : bcp années/hommes de travail	1
1.2 Motivation	1
1.2.1 Comment faciliter le dév ?	1
1.2.2 Prog haut niveau vs bas niveau	1
1.2.3 Scheme	1
1.3 Problématique	1
1.4 Méthodologie	2
1.4.1 Dev 1er jeu simple pour determiner les besoins pour Scheme	2
1.4.2 Augmenter Scheme pour repondre a ces besoins	2

1.4.3	Ecrire un nouveau jeu utilisant les techniques developpees	
	pour le 1er jeu . . . . .	2
<b>CHAPITRE 2 : DÉVELOPPEMENT DE JEUX VIDÉO . . . . .</b>		<b>3</b>
2.1	Contexte (intro ?) . . . . .	3
2.1.1	Grosueur du marche des JV . . . . .	3
2.1.2	Diversite des jeux (genres, plates-formes) . . . . .	3
2.1.3	Caractéristiques de jeux modernes . . . . .	3
2.2	Historique . . . . .	4
2.2.1	préhistoire 1948-1970 . . . . .	4
2.2.2	Système arcade 1970-1985 . . . . .	4
2.2.3	Premières consoles 1972-1984 . . . . .	4
2.2.4	Ordinateurs personnels 1977-... . . . .	4
2.2.5	Consoles portables 1980-... . . . .	4
2.2.6	Consoles intermédiaires 1984-2006 . . . . .	4
2.2.7	Consoles modernes 2005-... . . . .	4
2.3	Contraintes de programmation . . . . .	4
2.3.1	fluidité . . . . .	4
2.3.2	Modularité . . . . .	4
<b>CHAPITRE 3 : LE LANGAGE SCHEME . . . . .</b>		<b>5</b>
3.1	Programmation fonctionnelle . . . . .	6

3.1.1	Fonctions sont des données de premier ordre . . . . .	6
3.1.2	Fonctions d'ordres supérieures (avec exemples) . . . . .	6
3.1.3	Programmation fonctionnelle pure (exemples) . . . . .	6
3.1.4	Effets de bords dans Scheme . . . . .	6
3.2	Macros . . . . .	6
3.2.1	Introduction . . . . .	6
3.2.2	Exemples simples . . . . .	6
3.2.3	Problème de l'hygiène des macros . . . . .	6
3.2.4	Différentes formes spéciales (define-macro, define-syntax) . .	6
3.3	Continuations . . . . .	6
3.3.1	Introductions du sujet et explications . . . . .	6
3.3.2	Réification de continuations . . . . .	6
3.3.3	Exemples d'utilisations . . . . .	6
3.3.4	Exemple d'implantations . . . . .	6
3.3.5	Forme d'écriture de code en CPS . . . . .	6
3.4	Gestion mémoire automatique . . . . .	6
3.4.1	Motivation . . . . .	6
3.4.2	Historique et survol des techniques . . . . .	6
<b>CHAPITRE 4 : PROGRAMMATION ORIENTÉE OBJET . . . . .</b>		<b>7</b>
4.1	Motivation (ou Objectifs ?) . . . . .	8
4.1.1	Parler des define-type de Gambit-C et du SRFI-9 . . . . .	8

4.1.2	Parler de CLOS . . . . .	8
4.2	Description du langage . . . . .	8
4.2.1	Définition de classes . . . . .	8
4.2.2	Héritages des membres . . . . .	8
4.2.3	Définition de fonctions génériques . . . . .	8
4.3	Implantation . . . . .	8
4.3.1	Aperçu global . . . . .	8
4.3.2	Séparation entre le travail fait durant l'expansion macro et l'exécution . . . . .	8
4.3.3	Implantation de define-class . . . . .	8
4.3.4	Structures de données (descripteurs de classes, format des instances, etc..) . . . . .	8
4.3.5	Implantation de define-generic . . . . .	8
4.3.6	Implantation de define-method . . . . .	8
4.4	Conclusion . . . . .	8
4.4.1	Ouverture sur le fait qu'un meta-protocole serait très intéressant à ajouter, mais à quel prix? . . . . .	8
<b>CHAPITRE 5 : SYSTÈME DE COROUTINES . . . . .</b>		<b>9</b>
5.1	Motivation . . . . .	10
5.1.1	Contrôle fin du comportement de threads . . . . .	10
5.1.2	Parler de l'intérêt de Termite . . . . .	10



5.1.3	Motivation de l'utilisation de coroutines (contrôle exact sur le flot de contrôle == système toujours dans un état consistant).	10
5.2	Description du langage . . . . .	10
5.2.1	Création de coroutines . . . . .	10
5.2.2	Manipulation du flot de contrôle . . . . .	10
5.2.3	Système de communication inter coroutines . . . . .	10
5.2.4	Démarrage du système . . . . .	10
5.3	Implantation . . . . .	10
5.3.1	Implantation des coroutines . . . . .	10
5.3.2	Scheduler . . . . .	10
5.3.3	Système de messagerie . . . . .	10
5.4	Conclusion . . . . .	10
5.4.1	Ouverture sur le profilage des coroutine . . . . .	10
<b>CHAPITRE 6 : ÉVALUATION ET EXPÉRIENCES . . . . .</b>		<b>11</b>
6.1	Développement de « Space Invaders » . . . . .	12
6.1.1	Objectifs . . . . .	12
6.1.2	Version initiale . . . . .	12
6.1.3	Version orientée objet . . . . .	12
6.1.4	Version avec système de co-routine . . . . .	12
6.1.5	Conclusion . . . . .	12
6.2	Développement de « Lode Runner » . . . . .	12

6.2.1	Objectifs . . . . .	12
6.2.2	Synchronisation . . . . .	12
6.2.3	Machines à états . . . . .	12
6.2.4	Intelligence Artificielle . . . . .	12
6.2.5	Conclusion . . . . .	12
<b>CHAPITRE 7 : TRAVAUX RELIÉS . . . . .</b>		<b>13</b>
7.1	Comparaison de langages . . . . .	13
7.1.1	Lua . . . . .	13
7.1.2	C++ . . . . .	13
7.2	Jeux en Lisp . . . . .	14
7.2.1	QuantZ . . . . .	14
7.2.2	Naughty Dogz . . . . .	14
<b>CHAPITRE 8 : CONCLUSION . . . . .</b>		<b>15</b>
<b>BIBLIOGRAPHIE . . . . .</b>		<b>16</b>

## **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 développés en utilisant le langage de programmation Scheme. Le premier jeux a servi de plateforme 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 marché des JV**

##### **2.1.2 Diversité des jeux (genres, plates-formes)**

##### **2.1.3 Caractéristiques 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 venue 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]

### 3.1 Programmation fonctionnelle

#### 3.1.1 Fonctions sont des données de premier ordre

#### 3.1.2 Fonctions d'ordres supérieures (avec exemples)

#### 3.1.3 Programmation fonctionnelle pure (exemples)

#### 3.1.4 Effets de bords dans Scheme

### 3.2 Macros

#### 3.2.1 Introduction

3.2.1.1 Explications sur la simplicité provenant que le code scheme est  
utilise aussi comme donnée a un programme scheme (macro)  
comme un ASA.

3.2.1.2 Puissance de calcul lors de l'expansion.

#### 3.2.2 Exemples simples

#### 3.2.3 Problème de l'hygiène des macros

#### 3.2.4 Différentes formes spéciales (define-macro, define-syntax)

### 3.3 Continuations

#### 3.3.1 Introductions du sujet et explications

#### 3.3.2 Réification de continuations

#### 3.3.3 Exemples d'utilisations

#### 3.3.4 Exemple d'implantations

## **CHAPITRE 4**

### **PROGRAMMATION ORIENTÉE OBJET**

## 4.1 Motivation (ou Objectifs ?)

### 4.1.1 Parler des define-type de Gambit-C et du SRFI-9

### 4.1.2 Parler de CLOS

## 4.2 Description du langage

### 4.2.1 Définition de classes

#### 4.2.1.1 Compatibilité avec les define-type

#### 4.2.1.2 Définitions simples (instance slots et class slots)

### 4.2.2 Héritages des membres

#### 4.2.2.1 Utilisation de hook sur les slots

#### 4.2.2.2 Constructeurs

### 4.2.3 Définition de fonctions génériques

#### 4.2.3.1 Dispatch simple

#### 4.2.3.2 Dispatch multiple (avec les problèmes reliés à la résolution de la méthode à choisir)

#### 4.2.3.3 call-next-method

#### 4.2.3.4 Type '\*'

## 4.3 Implantation

### 4.3.1 Aperçu global

### 4.3.2 Séparation entre le travail fait durant l'expansion macro et l'exécution

## 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 potentiels

#### 6.1.2.2 Comment faire des animations ? =, CPS

#### 6.1.2.3 Comment concevoir une partie à 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 à 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://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.