Rapport final

M1IF 2010/2011

19 janvier 2011

Résumé

Ce document présente en détails le travail effectué par chaque groupe de travail pour le projet intégré 2010-2011.

Introduction

Cartomancer est un projet ambitieux : proposer un logiciel qui, à partir d'un ensemble de règles définies par l'utilisateur, génére un programme jouable sur ordinateur. Sans connaissance requise en programmation informatique bien sûr. C'est sur cette idée que la promotion de M1IF 2010-2011 a concentré ses efforts dans le cadre du projet intégré. L'étendue du sujet nécessitait une organisation fractionnée en petits groupes les plus indépendants les uns par rapport aux autres que possible afin de travailler efficacement. L'objectif était clair : une interface graphique où l'utilisateur pouvait définir facilement les règles de son jeu, presser un bouton pour l'enregistrer et une interface graphique pour y jouer ! Plusieurs points de vue ont été envisagés dès le début : devait-on créer notre propre langage et en créer un interpréteur ? Ou « Compiler » vers un langage comme Python ? Plusieurs idées sont apparues en même temps : un mode réseau bien développé et communautaire ? Un générateur automatique d'IA ? Un logiciel pour dessiner ses propres cartes (vectorielles !) ? Bref, Cartomancer est un sujet fécond et séduisant, sans cesse ouvert à de nouveaux développements. Sa richesse justifie notre choix et l'intêret que nous lui avons porté dès le début.

Le présent rapport tente, en cette fin de semestre, de donner un aperçu du travail qui a été effectué par chacun des groupes, de l'énorme travail qu'il reste encore à effectuer, des améliorations envisagées au cours de la création de Cartomancer et enfin de ce que nous avons tous retirés de cette expérience un peu particulière. Le document s'organise en plusieurs parties, chacune traitant du travail d'un groupe précis, rédigé par un de ses membres. On y retrouve donc un rapport du *WP Marketing* dont le travail était à la périphérie du logiciel en lui-même, mais indispensable. Ensuite vient le travaille du *WP Règle* qui a mis au point le langage de programmation décrivant les jeux de cartes. Puis le *WP GUI* qui s'est chargé de mettre au point une interface graphique offrant la possibilité de décrire les règles. Le *WP Interpréteur* suivi du *WP Réseau* expliquent comment interpréter un fichier de règle et les services réseaux proposés à l'utilisateur.

Table des matières

Première partie

Marketing

Chef de WP Jehanne Dousse

Membres Vincent Neiger, Boris Dalstein

Auteur de cette partie Jehanne Dousse

Le groupe de travail marketing s'est concentré sur les besoins périphériques du groupe. L'étude a été portée à la fois sur l'évaluation de l'intêret du logiciel auprès des utilisateurs ainsi que sur les moyens de distribution et de promotion de Cartomancer.

1 Travaux effectués

Le travail effectué par le groupe de travail Marketing est de nature très hétérogène : il couvre en effet à la fois la création du site internet de Cartomancer et des enquêtes sur des forums internet pour connaître les besoins des potentiels utilisateurs. Voici une liste de ce qui a été fait :

- *Création du site web* : du 10/10/2010 à la fin du projet (Vincent, Boris, Jehanne) Le site est accessible depuis l'adresse http ://graal.ens-lyon.fr/cartomancer. Le design a été amélioré depuis la première version. Le site a, durant une courte période, été à la fois en français et en anglais, mais nous avons préféré nous limiter finalement à la version anglaise car cela nous semblait suffisant . Le forum est maintenant intégré au site et une section utilisateur est présente.
- *Identité graphique* : fin octobre (Boris) Réalisation du logo de cartomancer et amélioration du graphisme du site web (cela rejoint un peu le premier point).
- *Étude des attentes des futurs utilisateurs* : de mi-octobre à début novembre (Jehanne, Vicent) Nous avons posté sur différents forums et recueilli les attentes des utilisateurs.
- *Aide à la création d'une documentation développeur* : vacances de la Toussaint (Vincent) Nous avons mis sur le wiki des liens vers de bons tutoriels concernant Doxygen.
- *Informations sur Cecill* : Vincent
- *Étude des logiciels existants* : vacances de la Toussaint (Jehanne et Boris) Test de Multimedia Fusion 2 et de Blender.
- *La présentation* : vacances de Noël (Jehanne) Création des slides de la présentation à partir des rapports fournis par chaque workpackage et organisation du déroulement de la présentation (temps de parole, choix des intervenants ...)
- *Rédaction des rapports du WP0* : Jehanne

2 Travail restant

Étant donné que le projet intégré n'a pas donné lieu a une version finale utilisable de Cartomancer, aucune documentation utilisateur ni tutoriel n'ont été écrit. Ce sera pourtant un travail indispensable avant de proposer le logiciel à une communauté d'internaute.

Deuxième partie

Règles

Chef de WP Martin Bodin

Membres Vincent Lanore, Sébastien Maulat, Thomas Piccetti, Julien Herrman, Jehanne Dousse, Sophie Bernard

Auteur de cette partie Martin Bodin

Cartomancer s'articule autour d'un fichier XML décrivant les règles d'un jeu de cartes. Ce fichier est généré par la partie GUI/IDE (éditeur de jeu de cartes) et lu par la partie Interpréteur/Client de jeu ; il peut également être utilisé par un utilisateur expérimenté pour écrire directement un jeu ou encore être généré/lu par d'autres logiciels qui voudraient s'interfacer avec Cartomancer.

Le rôle du groupe Règles est de mettre au point la syntaxe de ce fichier XML. Cette syntaxe doit permettre de décrire les règles de n'importe quel jeu de cartes « simple ».

3 Objectifs

3.1 Différents types de jeux de cartes

Deux grands types de jeux de cartes ont étés distingués : les jeux à combinaisons et les jeux à effets. La première catégorie regroupe les jeux où les règles décrivent les interactions entre les cartes (Belote, Poker, Tarot...) tandis que la seconde regroupe les jeux où les cartes décrivent l'effet qu'elles ont quand on les joue (Magic, Munchkin ...). La première catégorie regroupant la plupart des jeux « classiques » (mais aussi d'autres comme le UNO, Nothingham, Dobble...) il a été décidé de s'y limiter dans un premier temps. On doit donc pouvoir transcrire facilement en XML les règles des jeux à combinaisons.

3.2 Souplesse du langage

L'idée derrière ce fichier de règles est de pouvoir décrire facilement les règles d'un jeu de cartes. Ainsi, il a été décidé de se concentrer sur la facilité d'écriture pour un utilisateur humain au détriment de la facilité d'interprétation. Cela implique des concepts de « haut niveau » et la présence de sucre syntaxique. C'est là que se trouve la valeur ajoutée par les règles en XML par rapport à l'écriture d'un jeu dans un langage de programmation classique. Certes, il est assez difficile de comprendre le fichier XML, mais il reste plus simple à comprendre ou à modifier pour quelqu'un qui s'y connaîtrait plus en jeu qu'en programmation.

4 Les règles en XML

Le fichier de règles est divisé en deux grandes parties : la « déclaration » des objets de jeu (joueurs, cartes, zones de jeu) et la déclaration des règles elles-mêmes.

4.1 Déclaration des objets de jeu

On déclare le nombre de joueurs et les attributs des joueurs. Par exemple : *<new_player_limit min="2" max="6" />* pour spécifier qu'on décrit un jeu se jouant de 2 à 6 joueurs.

On déclare les zones de jeu (par exemple « centre de la table ») et pour chaque zone de jeu, les piles de cartes qui peuvent s'y trouver (par exemple « pioche »).

On déclare les différents types de cartes (par exemple « As de coeur », « 3 rouge »...) puis on déclare le ou les paquets que l'on va utiliser (par exemple, le jeu de 54 cartes classique).

4.2 Règles

Il a été décidé de séparer les règles en « états » (par exemple « étape de début de jeu », « fin de tour »...) avec des instructions permettant de passer d'un état à un autre. On a jugé que c'était une façon pratique et intuitive de découper les règles d'un jeu de cartes. Cette décomposition en états a été reprise par la partie GUI/IDE, mais à un niveau un peu plus haut : dans le fichier XML, les états sont généralement (à quelques exceptions *<can />* et *<wait />* près, voir le Wiki pour plus de précisions) non interruptibles, contrairement à ceux de l'éditeur graphique.

La partie règles du fichier XML est donc découpée en états. Chaque état est composé d'instructions. Les instructions peuvent être des instructions spécifiques aux jeux de cartes (« déplacer telle carte d'ici à ici ») ou des instructions d'un langage impératif classique (if, boucles for...). L'accent a été mis sur les instructions permettant de coder facilement les jeux de cartes les plus classiques. Les jeux un peu plus « originaux » demanderont parfois l'utilisation lourde d'instructions impératives classiques.

5 Avantages / défauts du langage, conclusion

Le groupe de travail a mis au point la norme 1.01 du langage de description de règles. C'est cette norme qui a été succinctement décrite ci-dessus. Il a également été proposé de nombreuses idées d'améliorations possibles du langage (baptisée norme 1.1) qui n'ont pas été incluses dans la norme 1.01 car leur implémentation (interpréteur) aurait pris trop de temps.

La langage est adapté aux jeux n'ayant pas recours à des mécanismes trop originaux. Ces jeux peuvent être codés très facilement et (presque) sans avoir recours à la partie « impérative classique » du langage. Un fichier de règles est assez proche de l'organisation des règles du jeu dans langage courant. On retrouve la description du contenu de la boîte, les grandes étapes de la partie, le nombre de joueurs ...

Cependant les jeux utilisant des mécanismes exotiques ou les jeux à effets sont pénibles à coder car les instructions de la norme actuelle ne couvrent pas suffisamment de jeux. Ce problème pourrait être réglé en spécifiant de nouvelles instructions (la norme 1.1 contient des suggestions). Le choix du XML produit des gros fichiers (balises) pas forcément très faciles à lire. Une façon d'écrire plus pratique pourrait être définie mais cela impliquerait la réalisation d'un parser plus complexe.

Troisième partie
GUI

Chef de WP Florent Capelli

Membres Martin Bodin, Étienne Miquey, Boris Dalstein, Sophie Bernard, Ioanna Domnina Cristescu

Auteurs de cette partie Florent Capelli, Boris Dalstein

6 Une interface simple

6.1 Démarche

Cartomancer a été conçu pour faciliter la création de jeux de carte sur ordinateur pour des personnes ne sachant pas ou ne voulant pas programmer quoique ce soit. Il n'est donc pas envisageable de ne laisser à disposition de l'utilisateur que le langage de script XML. L'effort qu'il devrait fournir pour l'apprendre serait équivalent à celui qu'il devrait fournir pour apprendre un langage de programmation normal ! Nous avons donc dirigé notre réflexion vers la conception d'une interface graphique simple à prendre en main qui exporterait ce que l'utilisateur aura créé vers le langage de script XML.

Notre démarche se fonde sur la constatation suivante : il est souvent très simple et rapide d'expliquer les règles d'un jeu de carte à un débutant mais elles sont beaucoup plus longues à formaliser ensuite dans un programme. Nous nous sommes donc penchés sur la façon dont on décrivait un jeu à l'oral pour essayer de traduire cela en une interface utilisable. Nous nous sommes aperçus que nous commencions à décrire le jeu en terme de *phases* avant de préciser ce qu'il se passait à l'intérieur de chacune. Ensuite, nous avons mis en lumière des schémas de jeu qui se reproduisent souvent et qui sont tellement naturels qu'on ne les explique pas à l'oral. Typiquement, on aura des phrases du genre : « chaque joueur *à son tour pose* une carte », « *à la fin du tour*, on ramasse les cartes », « lorsque les joueurs ont tous misé la même somme, on *distribue* deux cartes au centre » ...On ne trouve pas utile de préciser que les joueurs jouent à tour de rôle dans le sens des aiguilles d'une montre. De même, on retrouve des actions communes à tous les jeux de carte : « on pioche un carte », « on pose une carte » ...

6.2 Description de l'interface

L'interface se doit donc de proposer à l'utilisateur des abstractions simples de ces actions récurrentes tout en lui proposant aussi des outils proches de la programmation pour gérer les cas plus compliqués (on ne pourra pas se passer d'opérations comme les structures if-then-else par exemple. Cependant, on pourra faire un effort pour que leur utilisation soit le plus simple possible). Nous décrivons donc un jeu de carte en trois grandes étapes :

- l'utilisateur décrit ce qui concerne les joueurs : nombre de joueurs, attributs (les joueurs ont-ils de l'argent, des points, un statuts ?)
- l'utilisateur décrit ce qui concerne les cartes : deck à utiliser, attributs des cartes (couleur ? force ?). Cartomancer prévoit en plus un éditeur de carte basique pour que l'utilisateur puisse dessiner ses propres cartes.
- l'utilisateur décrit les règles à l'aide des *phases de jeu*. Chaque phase se déroule selon un *schéma de jeu*. Nous proposons alors des schémas de jeu simples comme le tour par tour. L'utilisateur décrit les actions du joueur courant, et cela s'appliquera à chaque joueur

chacun leur tour. De même, on propose un tour par tour avec une action finale. L'utilisateur décrit ici ce que les joueurs doivent faire chacun leur tour puis une action qui s'exécute une fois le tour fini !

On représente les phases de jeux et leurs transitions par un graphe et l'utilisateur n'a qu'à cliquer sur un état pour en éditer les propriétés. Pour illustrer cette théorie, nous allons désormais traiter un exemple complet : celui du tarot.

6.3 Un exemple complet

Nous traiterons l'exemple du tarot en deux grands mouvements : premièrement, nous allons reproduire la situation où nous serions amenés à expliquer les règles à un débutant. Ensuite nous verrons comment transcrire cela dans l'interface utilisateur !

D'abord, une transcription orale :

- *Distribuer* les cartes, en en mettant trois au chien
- *Faire un tour d'enchère*. On peut dire : « passe », « petite », « garde », « garde sans chien », « garde contre le chien ».
- Celui qui prend fait son chien. Il prend les trois cartes centrales et enlève trois carte de son jeu.
- Celui qui a pris commence. Il pose une carte, chaque joueur *à son tour* pose une autre carte de la même couleur. Si tu n'as pas de cette couleur, tu peux poser de l'atout. Si tu n'as pas d'atout non plus, tu peux poser n'importe quoi.
- *À la fin du tour*, on ramasse les cartes. Celui qui a l'atout le plus haut ou à défaut, la carte de couleur demandée la plus haute, remporte le pli.
- Celui qui remporte le pli recommence à jouer.
- Quand il n'y a plus de carte, on compte *les points*.

En général, après avoir défini ça, on commence à jouer et les détails sont réglés petit à petit avec le débutant. L'idée ici est que si on a bien défini nos phases de jeux, les détails se définiront rapidement à l'intérieur de chaque phase. Voir la figure 1 pour un exemple de ce qui pourrait être fait sous Cartomancer pour décrire ce jeu.

7 Une implantation modulable

7.1 Les objets manipulés

Actuellement, seuls le tour par tour, le tour par tour avec action finale et la distribution de carte sont implantés dans Cartomancer. Cependant, nous avons pensé l'interface pour que celle-ci soit très facilement modulable et nous permette d'ajouter rapidement de nouvelles commandes, de nouveaux schémas. Nous utilisons donc énormément l'héritage en C++ afin de ne pas redéfinir plusieurs fois les mêmes objets. Pour cela, nous définissons quelques grandes classes d'objets :

- Les structures de contrôle : typiquement les structures if-then-else, les while.
- Les instruction : ce sont des fonctions haut niveau pour la plupart, adaptées au jeu de carte. Par exemple : « piocher une carte », « argent(joueur) = argent(joueur)+10 » ...
- Les schémas de jeu, « Game Patterns ». Par exemple, « tour par tour ».
- Les phases de jeux, « Stage ». Ce sont les instanciations d'un game pattern. Par exemple, le tour par tour « enchères » dans le jeu de tarot.

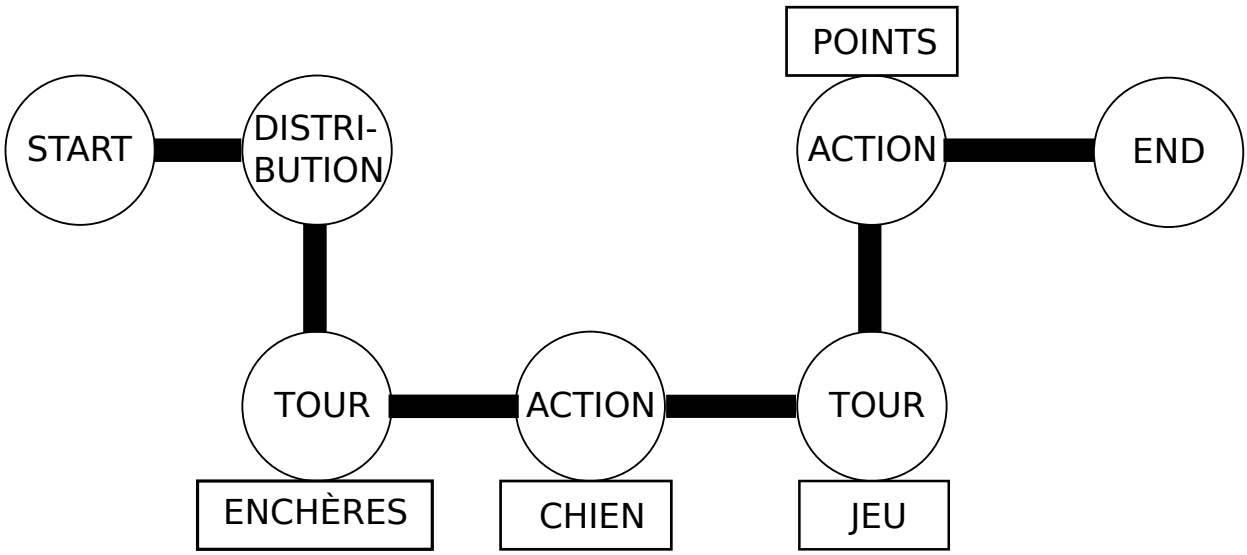


FIGURE 1 – Jeu de tarot sous Cartomancer

7.2 Un exemple : l'implémentation de « State »

Les structures de contrôle et les instructions peuvent être regroupées en un graphe « StateAutomaton ». Ils sont donc essentiellement de même nature. Ils dérivent donc tous deux d'une même classe « State », comme sur la figure 2. Cette vision générale des objets permets une implémentation plus rapide des différentes primitives. En effet, il suffit de créer une nouvelle classe par fonction.

Supposons qu'on veuille implémenter une fonction « couper » qui coupe un tas de carte en deux et met la partie basse sur la partie haute. C'est une instruction. On crée donc une classe « State_cut » qui dérive de « StateInstruction » et qui ne spécifie que ses propriétés ainsi que les données graphiques pour les éditer. Ici, on dira qu'on a un champ déroulant où l'utilisateur pourra sélectionner un des tas de carte présent dans le jeu, tas de carte qui sera coupé. Il faudra ensuite spécifier comment cette commande est traduite vers le XML.

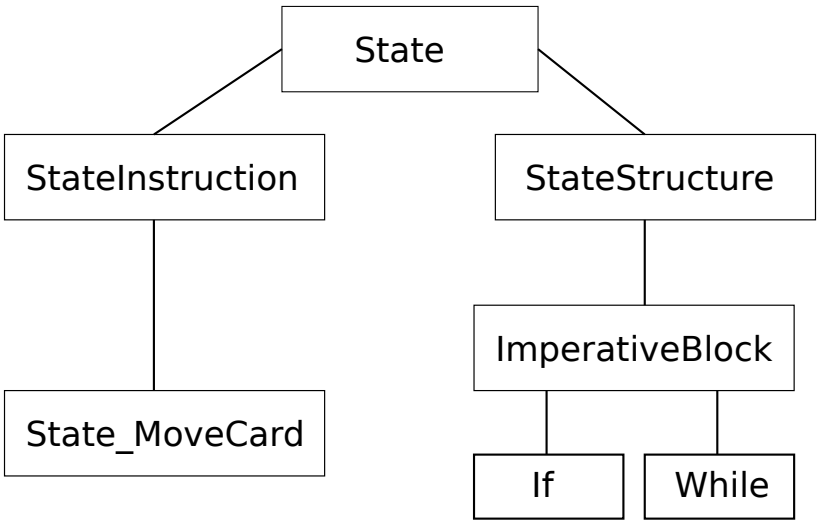


FIGURE 2 – Héritage de la classe abstraire State

8 Description détaillée des classes

8.1 Définir les règles du jeu : Implémentation des concepts

Afin de pouvoir tout faire fonctionner, il fallait donc créer une structure de donnée adéquate, ainsi que l'interface graphique permettant de manipuler cette structure de donnée. Les figures suivantes représentent la hiérachie en mémoire de Cartomancer lors de son exécution. Un flèche signifie qu'il y a un pointeur d'un objet vers un autre.

Il y a donc en pratique deux types d'objets : les objets de données (représentés ici par des rectangles), et les objets graphiques manipulant ces données (représentés par des ellipses). Dans le cas présent, la structure de donnée est quelque chose de similaire à des graphes imbriqués.

8.2 Automate des phases de jeu

Le graphe de plus haut niveau est un *StageAutomaton*, et le widget graphique le manipulant est *StageAutomatonWidget* (voir figure 3). On peut ainsi, via des boutons et la représentation graphique de l'automate (*StageAutomatonGraphicsScene*), créer et manipuler des *Stage*s, ainsi que créer les liens entre eux pour former l'automate. La représentation graphique d'un *Stage* est un objet manipulable à la souris, nommé *StageGraphicsItem* qui possède un pointeur vers le *Stage* correspondant, pour modifier les données suite aux interactions de l'utilisateur avec l'objet graphique.

Cependant, en plus de créer des *Stage*s et de les connecter entre eux, il est nécéssaire de personnaliser chaque *Stage*. Pour cela, chaque instance de la classe *Stage* possède, en plus de ses données/propriétés internes, un pointeur vers le widget graphique permettant de manipuler ces propriétés (*StagePropertiesWidget*, cf figure 4 pour plus de détails). Lorsque, via *StageAutomatonWidget*, l'utilisateur sélectionne un *Stage*, Cartomancer en est informé par un signal, et peut donc afficher, dans un cadre situé à gauche, le *StagePropertiesWidget* permettant de modifier ce "Stage".

8.3 Automates des états (pour une phase donnée)

La figure 4 détaille les données internes dun *Stage*, ainsi que *StagePropertiesWidget* qui permet de les manipuler. Les données d'un *Stage* sont un "GamePatern" (le schéma de jeu utilisé), ainsi que plusieurs *StateAutomaton*, qui instancient ce schéma de jeu (Que doit faire chaque joueur lorsque c'est son tour ? Que doit-on faire à la fin, lorsque chaque joueur a fini son tour ? etc...). Ainsi, à l'instar du graphe de haut niveau où on avait un automate *StageAutomaton* et un widget "StageAutomatonWidget", on a ici N automates *StateAutomaton* et N widget "StateAutomatonWidget" pour manipuler ces automates. La structure est ainsi proche de celle de la figure 3.

De plus, de la même manière que chaque *Stage* posséde un pointeur vers un *StagePropertiesWidget* pour modifier ses données internes, chaque *State* possède un pointeur vers un "StatePropertiesWidget" (cf figure 5). La sélection d'un *State* est alors informée au *StagePropertiesWidget*, qui peut ainsi afficher les propriétés de ce *State*, et permettre à l'utilisateur de les modifier.

8.4 Schémas de jeu

Un *GamePattern* est lui aussi un automate, très proche de l'automate des états. En fait, c'est même exactement un *StateAutomaton*, dans lequel on a accés en plus à un état "State_InnerAutomaton",

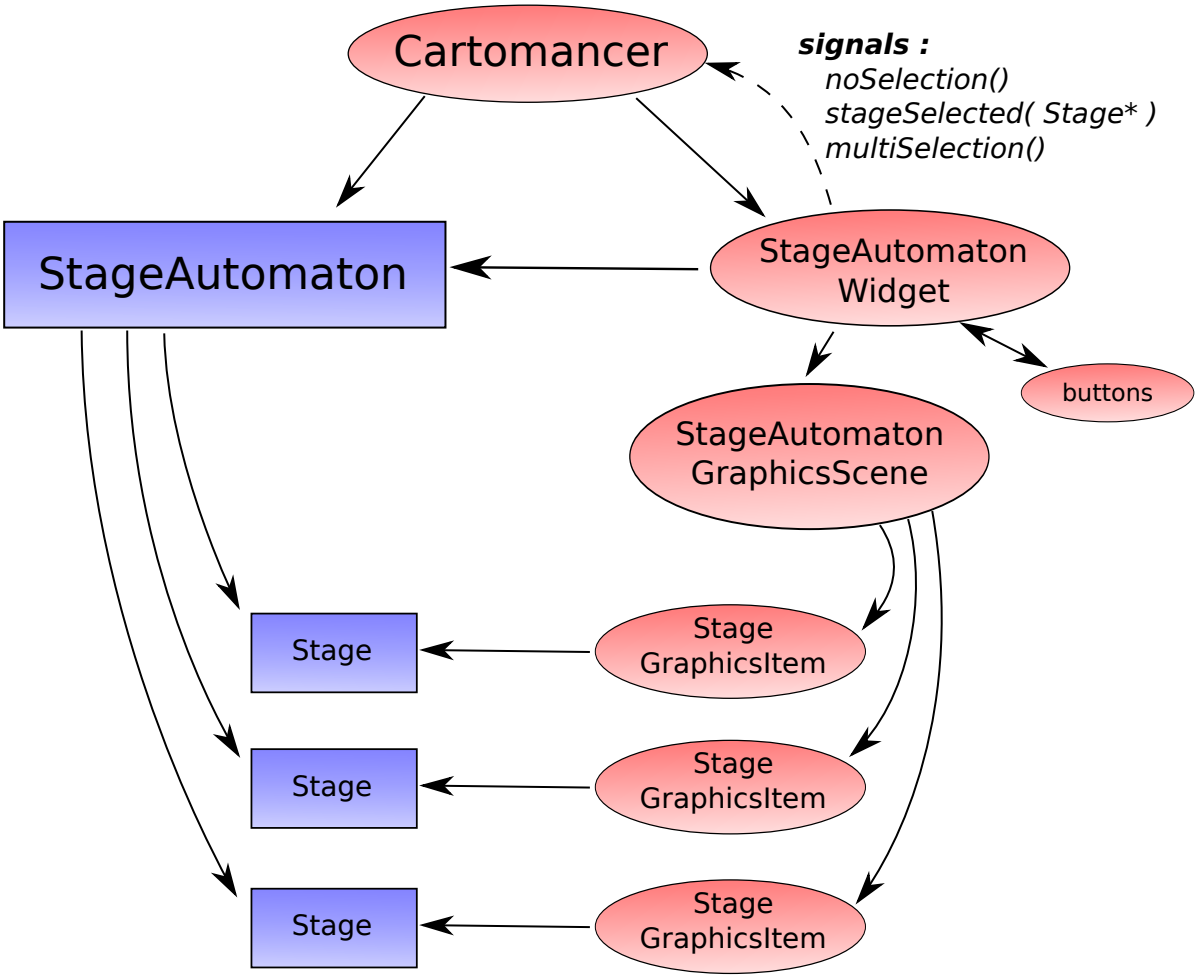


FIGURE 3 – Structure de donnée de l'automate des phases

qui est un état signifiant : "Lorsque l'utilisateur créera une phase de jeu suivant ce schéma de jeu, il faudra qu'il crée un automate qui s'insèrera à la place de ce macro-état"

Le nombre de "State_InnerAutomaton" d'un schéma de jeu correspond donc à l'entier N précisé dans le paragraphe précédent. (sur la figure 4, on a N=2).

Cartomancer permet à l'utilisateur, via l'onglet *PatternArea* de modifier les schémas de jeu existant ou de créer ses propres schémas de jeu. La figure 6 en décrit l'implémentation. Un schéma de jeu n'étant en fait rien d'autre qu'un *StateAutomaton* avec un type d'état de plus, sa structure de donnée est bien évidemment une sous-partie de celle représentée figure 4.

8.5 Conclusion

La structure de donnée choisie est une implémentation efficace des concepts de Cartomancer décrits précédemment. Les différents modules graphiques permettent de créer et d'agir intuitivement et rapidement sur les règles de jeu.

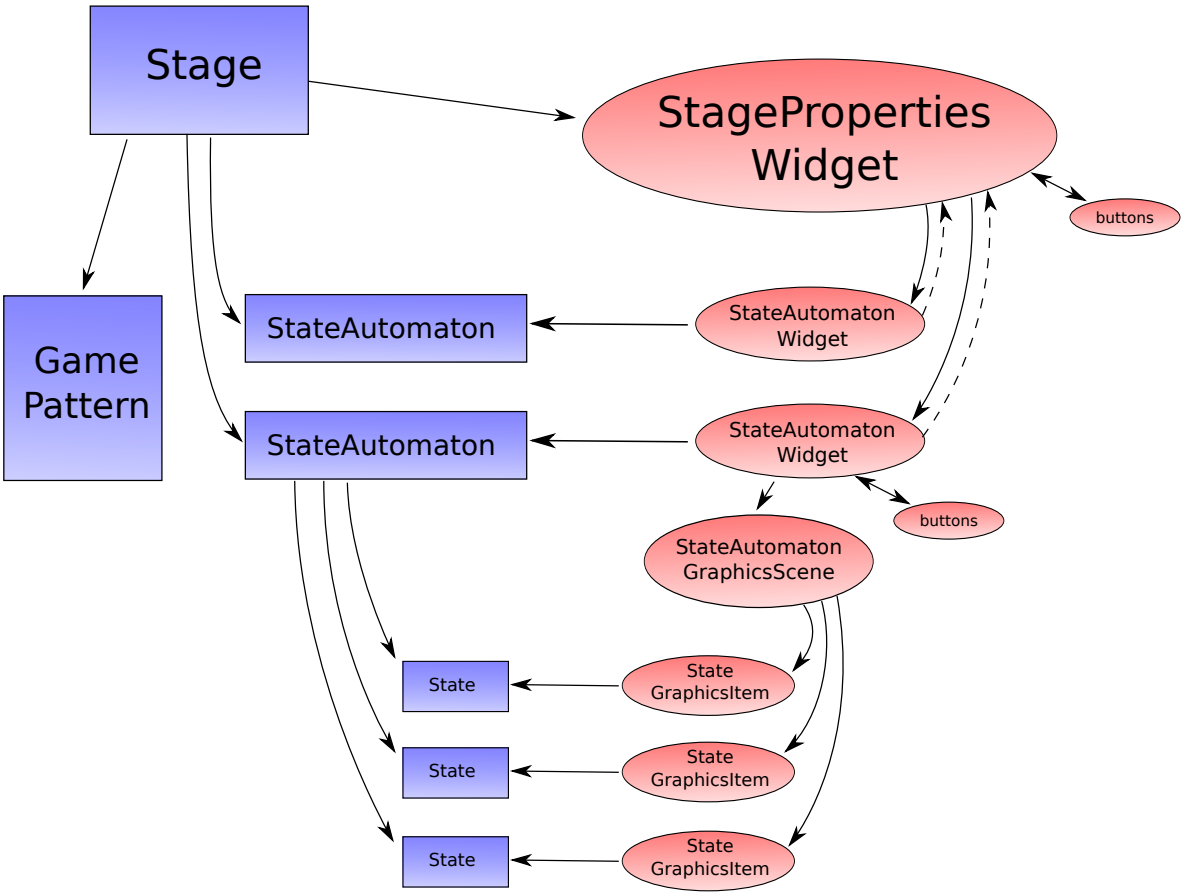


FIGURE 4 – Structure de donnée d'un automate des états

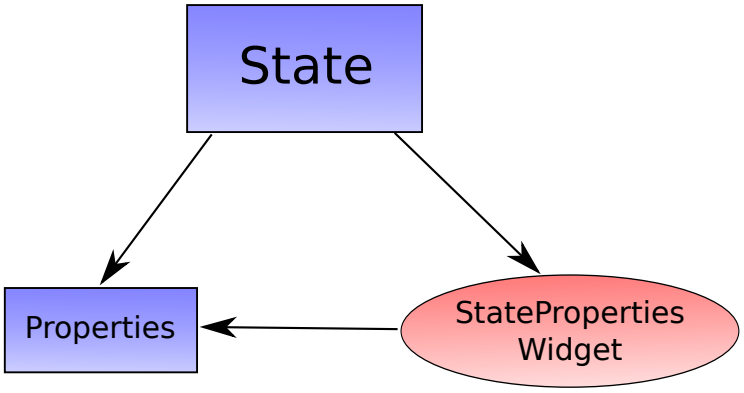


FIGURE 5 – Structure de donnée d'un état

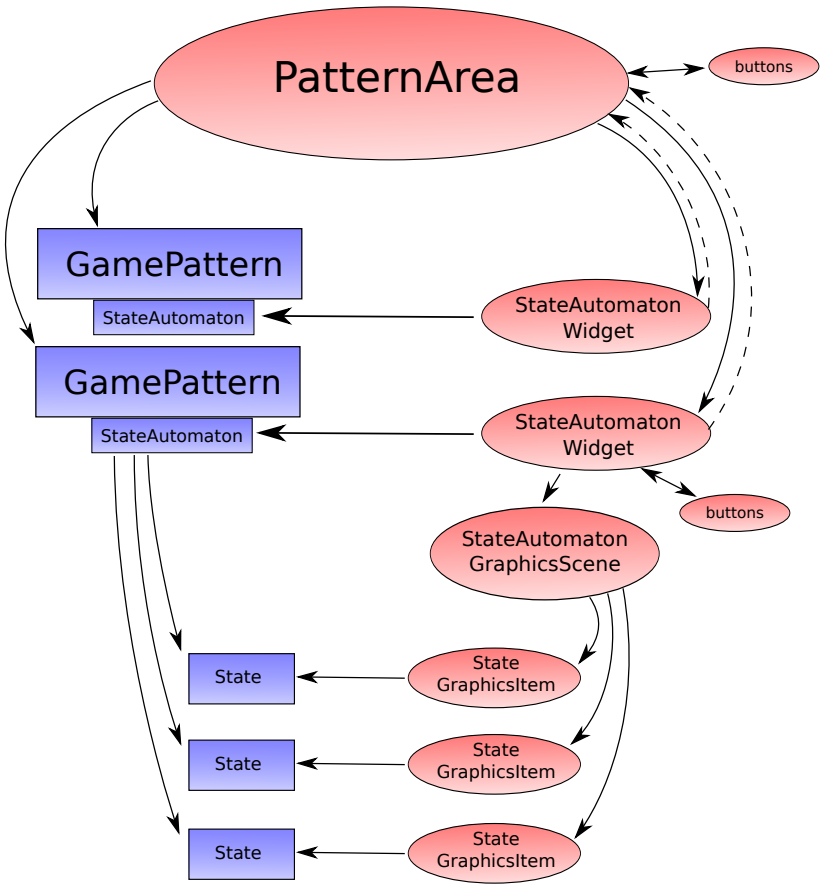


FIGURE 6 – Structure de donnée des schémas de jeux

Quatrième partie

Interpréteur

Chef de WP Vincent Lanore

Membres François Gindraud, Jérémie Dumas, Boris Dalstein, Ioanna Domnina Critescu, Alexandre Isoard

Auteur de cette partie Jérémie Dumas

Le choix d'utiliser un fichier de règle XML afin de décrire un jeu de carte comporte de nombreux avantages pratiques et permet d'ouvrir un peu plus le projet Cartomancer à une communauté de programmeurs qui pourraient coder des alternatives en amont ou en aval de ce fichier de règles. C'est pourquoi nous avons besoin d'un interpréteur. Le but de ce groupe de travail est clairement de mettre au point un interpréteur efficace du fichier XML, qui permettrait de jouer à un jeu créé.

9 Parsing

Un module s'occupe de parser et de lire les fichiers de jeu. Pour l'instant il s'agit de deux fichiers `cards.xml` et `rules.xml`. Le fichier `cards.xml` contient la base de données relative au design des cartes, telle qu'exportée par le *Card Editor* : on associe à chaque nom de carte une image (svg ou png, peu importe) pour la face de la carte, et une image pour le dos de la carte.

Le fichier `rules.xml` est celui défini par le *WP Règles*, il contient donc la description des différentes étapes du jeu. Le parseur s'occupe alors de lire ce fichier, d'en déduire la liste des cartes, piles, zones, variables, joueurs etc. nécessaires. Il crée ces différents éléments, et instancie également les états de l'automate correspondant aux différentes instructions et expressions qui sont lues.

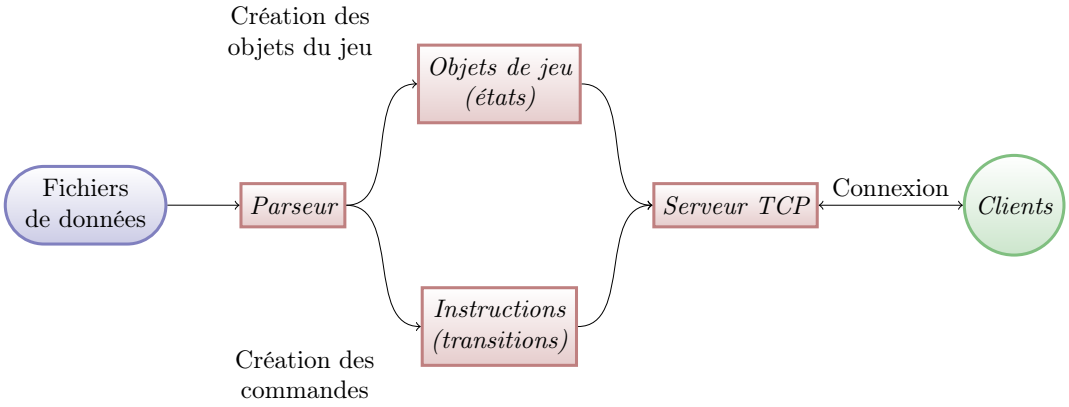
10 Network

Une partie du serveur écoute évidemment les connexions réseaux. Quand on démarre le serveur en lui fournissant un jeu à lancer, le parseur récupère l'information sur le nombre de joueurs attendus tel qu'indiqué dans le fichier `rules.xml`, ensuite on attend que suffisamment de clients se soient connectés. À ce moment là, l'utilisateur qui a lancé le serveur peut décider de commencer à jouer ou pas.

Une fois le jeu lancé, le parseur va lire la totalité du fichier de règles et instancier les différents objets en conséquence. Après quoi il lit les différentes instructions et expressions du fichier de règles et instancie de même les éléments de l'automate qui vont bien. Enfin, le jeu peut commencer et le serveur communique les info' nécessaires au client pour qu'il puisse jouer (se référer au compte-rendu du réseau pour plus de détails).

11 Objets de jeu

Le serveur dispose de plusieurs classes représentant les différents concepts introduits dans le XML par le *WP Règles*. On représente donc les différents objets nécessaires à la simulation d'un jeu à l'aide de ces classes. Cela consistue en fait les états d'un gros automate implicite qui se cache derrière le programme de simulation.


Création des
objets du jeu
Objets de jeu
(états)
Fichiers
de données
Parseur
Serveur TCP
Connexion
Clients
Instructions
(transitions)
Création des
commandes

12 Automate abstrait

Le serveur dispose également d'un automate abstrait pour exécuter les différentes instructions décrites dans le fichier XML des règles. Cet automate assez général réimplémente une bonne partie de la norme POSIX, et gèrera les problèmes d'accès concurrentiels et d'exécution parallèle créés par le fichier XML. En effet la description des règles faites dans le fichier XML est assez haut niveau dans le sens où on dispose de balise pour décrire des actions qui peuvent se faire de manière non séquentielle.

Cinquième partie

Réseau

Chef de WP Duco van Amstel

Membres Alexandre Isoard, Duco van Amstel

Auteur de cette partie Duco van Amstel

Le WP Réseau vise à ouvrir les jeux produits par Cartomancer à un cadre multi-joueur. Que cela soit sur un simple réseau local de type LAN ou par l'intermédiaire du réseau internet, Cartomancer doit permettre à des joueurs de différents pays de partager des sessions de leur jeu favori.

13 Les spécifications

Afin de s'insérer dans le travail global du projet notre travail a dû respecter certaines conditions imposés par les autres workpackages. Nous rappelons que notre logiciel est réalisé avec la librairie Qt ce qui nous permet de tirer un grand avantage de l'utilisation du système de signaux propre à Qt.

Nous devions donc fournir pour le serveur interpréteur du jeu :

- Une interface simple pour l'utilisation du WP Interpréteur comprenant :
  - Une fonction d'envoi de message *sendMessage()*.
  - Un signal *newMessage()* envoyé à chaque message reçu contenant le message.
- La capacité de vérifier que chaque client possède les fichiers nécessaires au bon déroulement du jeu.
- La capacité de transférer les fichiers manquants au client lorsque cela s'avère nécessaire.
- Une forme d'authentification des clients permettant de vérifier l'identité des clients.
- La gestion des nouvelles connexions, déconnexions involontaires et reconnexions.

A ceci s'ajoute de la même façon les spécifications pour le client, soient :

- Une interface simple pour l'utilisation du WP Interpréteur comprenant :
  - Une fonction d'envoi de message *sendMessage()*.
  - Un signal *newMessage()* envoyé à chaque message reçu contenant le message.
- Les fonctions implémentant la vérification et le transfert des fichiers par le serveur.
- Les fonctions implémentant l'authentification auprès du serveur.
- La connexion à un serveur de jeu et la reconnexion en cas de déconnexion involontaire.

Avec ces spécifications connues nous pouvons maintenant présenter l'architecture du serveur suivie de celle du client que nous avons mis en place.

14 Architecture réseau du serveur

Le serveur possède deux composantes réseaux distinctes.

14.1 Gestion des connexions à l'échelle du jeu entier

Lors de la création d'une partie avec le lancement du serveur du jeu celui-ci active un socket TCP qui écoute les interfaces réseaux de l'ordinateur pour détecter les connexions de nouveaux

joueur. Ceci se base sur l'utilisation d'un objet de classe *QTcpServer*. Le port d'écoute par défaut est le 4218 mais il peut-être spécifié lors du lancement du serveur par un joueur.

Une fois qu'une nouvelle connexion est obtenue sous la forme d'un *QTcpSocket* il est demandé au client s'il a déjà été connecté au serveur par le passé (dans le cas d'une reconnexion) et si oui de fournir sa clé d'authentification.

Celle-ci est ensuite comparée à celles connues par le serveur afin de trouver à quel joueur déconnecté correspond cette nouvelle connexion. Une fois identifié le socket correspondant au joueur en question est mis à jour. Si aucune clé ne correspond, la nouvelle connexion est rejetée. Si le joueur n'a jamais été connecté le serveur vérifie si de nouveaux joueurs sont admis à rejoindre le jeu et si tel est le cas, les objets des classes qui contiendront les informations de jeu relative à ce joueur sont instantiés. Sinon la connexion est rejetée.

14.2 Gestion de la communication avec un joueur spécifique

Chaque joueur connecté au serveur possède sa propre instance d'une classe *NetworkPlayer* (le communicateur) qui sert d'interface de communication pour le serveur avec ce joueur spécifique. De même chaque client possède un communicateur connecté au serveur.

Lors de la première connexion le communicateur fournit une clé d'authentification aléatoire au client valable pour le restant du jeu. Le client doit confirmer la clé en la renvoyant avant que le joueur ne soit considéré comme présent. Ensuite vient le contrôle des fichiers nécessaires à l'exécution du jeu. Le dossier où sont stockés les scripts de Cartomancer est défini par défaut comme le dossier *gamescripts* dans le dossier où est installé Cartomancer. La fonctionnalité de pouvoir choisir un dossier quelconque sera disponible dans une prochaine version puisque la structure sous-jacente permet cette manipulation.

Le résultat du contrôle des fichiers est la copie exacte de l'arbre des fichiers correspondant au jeu. Les fichiers existants du côté client sont vérifiés pour leur correction par leur hash de la fonction MD4 envoyé par le serveur. Si un fichier n'existe pas ou si le hash ne correspond pas, l'éventuel fichier corrompu est détruit et la version du serveur est téléchargée.

Les messages qui sont échangés entre le serveur et le client peuvent être de deux types : service ou contenu. Le premier est le type des messages qui sont gérés à l'intérieur du communicateur, le deuxième est le type des messages que le communicateur transmet à l'aide du signal *newMessage()* à l'interpréteur, que ce soit chez le client ou sur le serveur.

15 Conduite du projet

15.1 Etat du WP Réseau à la date de soumission du délivrable final

A la date limite de soumission du délivrable l'avancement des travaux du workpackage est le suivant :

Travail effectué

- Communication de messages entre serveur et client
- Authentification des clients
- Gestion des nouvelles connexions et des reconnexions
- Vérification complète des fichiers de script du jeu
- Transfert des fichiers absents ou corrompus
- Vérification et test des fonctionnalités proposées

Travaux futurs et en cours

- Une refonte de la structure des classes et objets du module réseau - *en cours de réalisation*
- La possibilité de crypter et sécuriser les connexions par l'intermédiaire du protocole SSL - *en cours de réalisation*
- Un serveur annuaire de jeux proposés par les joueurs tel que cela avait été prévu initialement dans le compte-rendu de mi-chemin - *définition des spécifications*
- Une interface graphique pour le client pour la connexion au serveur annuaire
- La possibilité de partager ses scripts sans joueur par l'intermédiaire du serveur annuaire

16 Conclusion

À la fin de la période de développement initiale nous pouvons constater que le WP Réseau a su fournir au projet une interface complète et fonctionnelle qui permettra à Cartomancer de faire abstraction de la localisation du serveur et des différents joueurs participant au jeu. Cette interface est cependant limitée et nécessite encore des améliorations tels que celles données dans les travaux futurs envisageables.

Malheureusement l'idée du serveur annuaire n'a pas abouti à des résultats plus concrets faute de temps lors de la phase finale du développement. Néanmoins le concept sera repris par la suite dans les travaux à venir et devra avec le temps apporter un outil efficace pour Cartomancer et pour la communauté des joueurs qui permettra, espérons-le, de toucher un public plus grand de joueurs potentiels.

Conclusion

Cartomancer est loin d'être fini. Aucune des parties du logiciel ne sont encore réunies et aucun jeu, même programmé à la main, n'est complètement interprété. Cependant, les bases théoriques sont solides et les principales idées avancées peuvent être facilement implémentées grâce à l'organisation du code par programmation objet. Le projet était certainement trop ambitieux pour une durée si courte. Il aura fallu du temps pour que le langage de règle soit suffisamment mature pour être utilisé, et donc pour que les autres groupes commencent à travailler, ce qui explique la solidité de la théorie en comparaison à l'implémentation de Cartomancer. L'avenir de Cartomancer a été discuté sur la mailing list et il semblerait qu'il sera retiré de graal pour migrer vers un svn sur SourceForge (http ://sourceforge.net/projects/cartomancer/) où son développement pourra continuer plus tranquillement cette fois-ci, jusqu'à avoir une première version utilisable.