Projet de programmation
-
Cahier des besoins
-
Mise en place du "Le Jeu de la Guerre" écrit par Guy DEBORD

Émile BARJOU, Christophe CAUBET, Adrien HALNAUT, Romain ORDONEZ

2 février 2018

Résumé

Le *Jeu de la Guerre*[5] est un jeu de plateau pensé par Guy Debord. C'est un jeu au tour par tour qui est centré sur la stratégie militaire et les contraintes auxquelles font face les armées en temps de guerre. Il oppose deux armées ayant pour objectif de détruire les troupes adverses ou ses moyens de communication.

Guy Debord s'est inspiré du jeu *Kriegsspiel*, crée par Von Reiswitz au XIXème siècle, qui permet de simuler une guerre entre deux armées permettant ainsi d'affiner et de prévoir les tactiques militaires et les affrontements, tout en s'initiant à l'art de la guerre. Ce jeu a été utilisé pendant des décennies par des officiers et commandants, d'abord allemands puis de diverses nationalités, afin de pratiquer des batailles et de prévoir des stratégies d'action. Guy Debord s'est également inspiré de plusieurs des ouvrages de Carl von Clausewitz, dont le plus connu, *De la guerre* (ou *vom Kriege*)[3], écrits à la même époque qu'a été crée le Kriegsspiel de von Reiswitz[2].

Ce sujet nous a été proposé par Philippe Narbel, avec qui nous avons pu préciser les besoins autour du projet. Son but sera principalement d'implémenter une version jouable du *Jeu de la Guerre*.

Il s'agira tout d'abord de comprendre suffisamment les règles du jeu pour pouvoir les implémenter dans le moteur de jeu puis d'ajouter une interface pour rendre le jeu jouable. Enfin, nous devrons extraire des informations pertinentes du jeu qui puisse être utilisées pour élaborer des stratégies.

1

Description et analyse de l'existant

1.1 Le Jeu de la Guerre

Le projet se basant sur le livre écrit par Guy Debord (voir [5], pages 135-160), il s'agit donc de notre source principale. Dans celui-ci est raconté le déroulement complet d'une partie, ainsi que la liste des règles :

1.1.1 Les éléments du plateau

Le plateau fait 25 cases de long, et 20 cases de hauteur, divisé en deux parties égales par une ligne sur le sens de la longueur. Chacune de ces parties constitue l'un des deux camps adverses et possède 3 cases forteresses, 2 arsenaux, 9 montagnes et 1 col placé entre deux cases montagnes. Ces éléments ne peuvent se superposer sur une même case.

Les montagnes ne peuvent pas être occupées ou traversées par une unité, et une attaque ne peut être portée à travers celles-ci.

Les forteresses, ne pouvant être détruites, ainsi que les cols apportent un bonus défensif aux unités les occupant sans se soucier du côté du terrain où ils sont situés.

Un arsenal est considéré comme détruit lorsqu'une unité ennemie a occupé sa case. Si une unité adverse se trouve sur une case arsenal lui appartenant, cette unité doit d'abord être détruite avant de pouvoir s'y déplacer.

Chaque camp possède également plusieurs types d'unités, définis par 4 caractéristiques : le coefficient d'attaque, le coefficient de défense, la portée et les points de mouvement.

1.1.2 Initialisation de la partie

Les joueurs devront disposer ensemble les montagnes et les cols sur le plateau, en respectant les contraintes précédentes.

Chacun leur tour, les joueurs devront placer leurs bâtiments et leurs unités sur le terrain sans avoir connaissance du placement des effectifs adverses. Les forteresses doivent être posées sur des cases en communication directe (voir partie sur les communications) avec au moins un arsenal allié.

Le joueur qui commence la partie est désigné de manière aléatoire.

1.1.3 Déroulement d'un tour

À son tour, un joueur peut déplacer jusqu'à 5 de ses unités. Une unité ne peut pas se déplacer plus d'une fois par tour, et elle ne peut se déplacer, attaquer ou défendre que si elle est alimentée en communication. À l'issue de chaque mouvement, le joueur peut attaquer une unité ennemie à portée d'attaque de l'unité déplacée.

1.1.4 Conditions de victoire

Il existe deux moyens de remporter la victoire, soit avoir détruit les deux arsenaux de l'adversaire, soit avoir détruit toutes ses unités combattantes.

1.1.5 Les communications

Les cases-Arsenal diffusent les lignes de communication dans les 8 directions (horizontales, verticales et diagonales), sans limitation de portée. La diffusion dans l'une des directions est stoppée dès lors qu'elle croise une unité ennemie combattante ou une montagne.

Les unités non-combattantes (les unités relais) alimentées en communication diffusent l'information de la même manière que les arsenaux. Les unités combattantes alimentées en communication diffusent l'information sur leurs 8 cases adjacentes, ainsi les unités se trouvant sur ces cases diffusent à leur tour l'information.

1.1.6 Attaquer et défendre

- L'attaque se déroule sur une case définie par le joueur attaquant, à savoir la case où se trouve l'unité attaquée.
- Toutes les unités du joueur à portée d'attaquer cette même case additionnent leurs coefficients offensifs. Les unités du joueur défendant à portée d'attaquer cette même case additionnent leurs coefficients défensifs.
- La différence entre le total des coefficients offensifs et le total des coefficients défensifs permet trois situations :
  1. La différence est inférieure ou égale à 0, il ne se passe rien.
  2. La différence est supérieure ou égale à 2, l'unité sur la case ciblée est détruite.
  3. La différence est égale à 1, lors du prochain tour, l'adversaire devra en premier utiliser un de ses coups pour déplacer l'unité sur la case ciblée. Cette unité ne pourra participer à aucune offensive pendant ce prochain tour.
- Une unité ayant attaqué ne peut plus se déplacer jusqu'à la fin du tour.
- Une unité de cavalerie peut, à l'issue de son déplacement, initier une charge contre une unité ennemie se trouvant sur une des 8 cases adjacentes à une unité de cavalerie. Cette unité peut être la même que celle qui initie l'attaque, et doit se trouver sur l'axe reliant l'unité attaquée et celle attaquante sans discontinuité de cases entre elles. La charge donne un coefficient offensif de 7 à cette unité de cavalerie, et permet de faire participer à l'offensive les autres unités de cavalerie alliées se trouvant dans le même alignement de cases sans discontinuité de cases entre elles. Ainsi les unités de cavalerie participant à la charge voient leurs coefficients offensifs montés à 7.
- Une charge ne peut être initiée contre une unité se situant sur un col de montagne.

1.2 Les autres projets

Il existe d'autres projets autour du *Jeu de la Guerre*, l'un d'entre eux venant d'un ancien projet de programmation dans le cadre de cette UE.

PdP 2014[4]

Ce projet permet la visualisation d'une instance d'une partie sans interaction possible avec le plateau. Ainsi, l'interface permet la visualisation des lignes de communications directes, mais pas indirectes, la portée de déplacement d'une unité ainsi que le potentiel offensif et défensif de chaque camp sur chaque case. Ce projet pourra nous donner une idée de fonctionnalités à implémenter si on veut développer une interface graphique pour l'utilisateur.

Kriegspiel par RSG[1]

Ce projet n'est plus disponible sur le site de Radical Software Games, mais les sources restent cependant trouvables sur d'autres sites tiers. Il permettait de jouer en réseau contre un autre joueur au *Jeu de la Guerre*, cependant des règles ont été simplifiées. Des aides visuelles sont présentes sur le plateau du jeu, permettant de voir les lignes de communication de chaque camp, ainsi que les cases où peut se déplacer une unité et si ces cases sont à portée d'une ligne de communication ou non. Il semble bien fonctionnel, mais il a été retiré du site pour une refonte, et semble malheureusement abandonné depuis.

FIGURE 1.1 – Capture d'écran du programme développé du PdP 2014

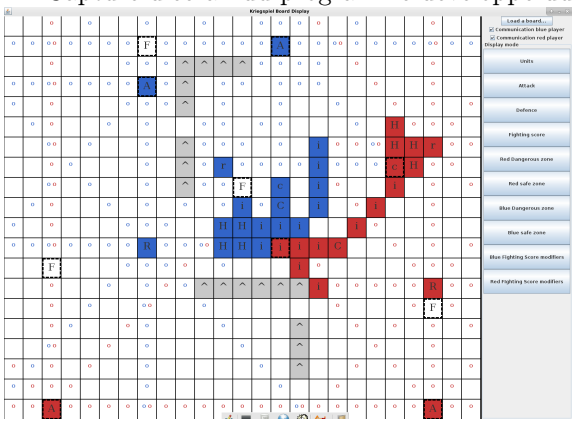


FIGURE 1.2 – Potentiel d'attaque des unités rouges

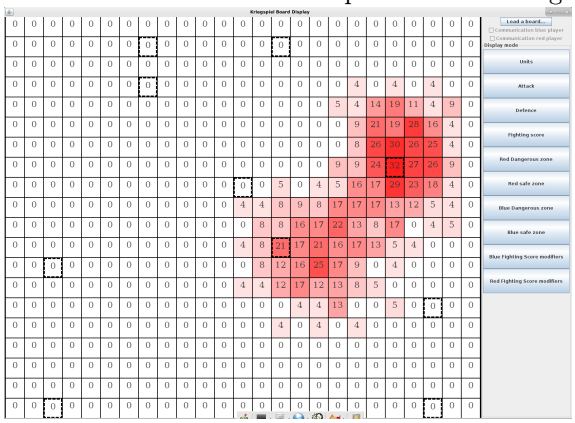
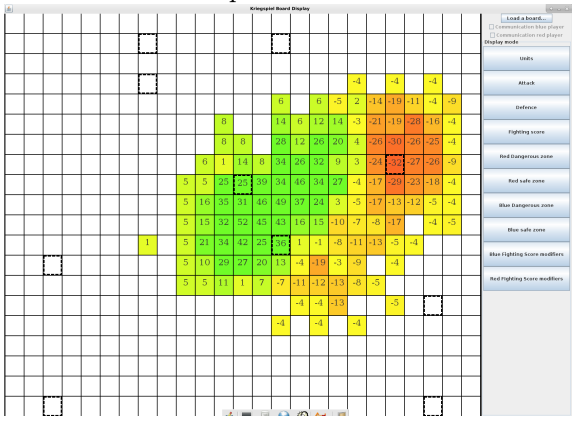


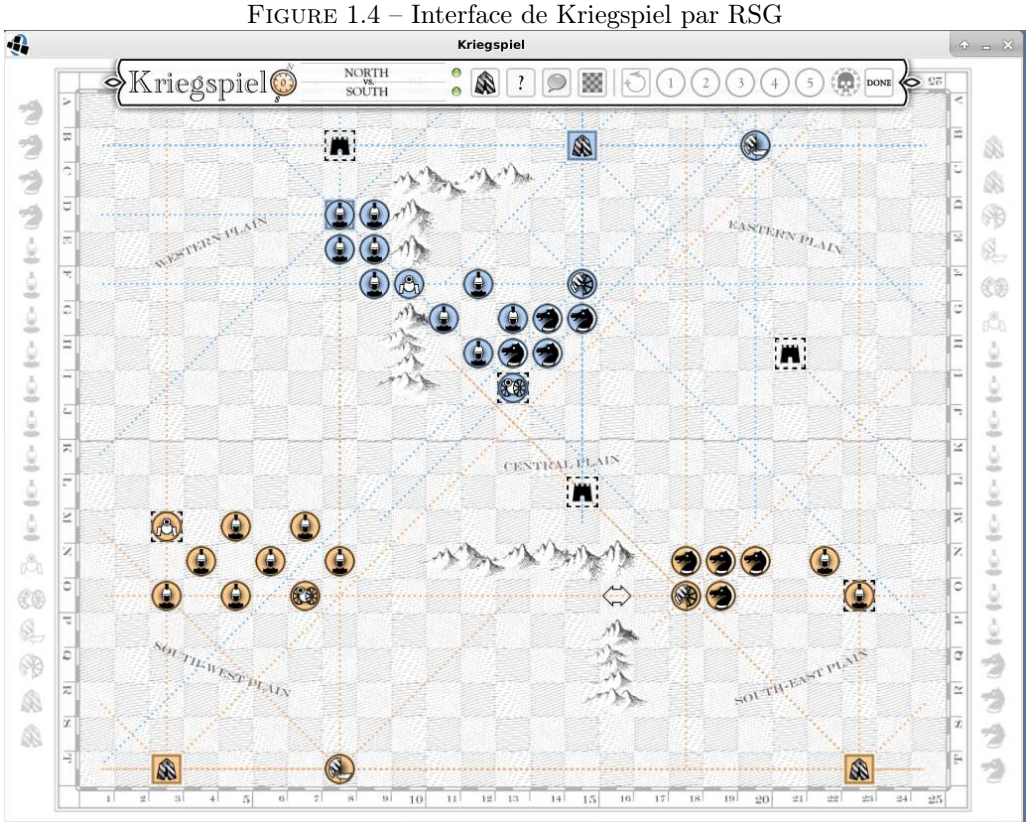
FIGURE 1.3 – Variation de potentiel de défense des unités bleues



Conclusion

À notre connaissance, et après des recherches, nous pensons donc qu'il n'existe pas de version jouable du *Jeu de la Guerre* actuellement disponible. Cependant il existe d'autres jeux semblables au jeu de Debord partageant le principe de simuler une guerre sur un plateau afin de revivre des batailles historiques et/ou de comprendre les tactiques derrière ces affrontements. On peut notamment citer *DesertFox*[8] qui était à l'origine le jeu proposé pour ce sujet mais d'une complexité bien supérieure à celle du *Jeu de la Guerre*. Nous devrons donc recréer tout du début, mais les projets que nous avons cités pourront tout de même nous servir dans notre réflexion sur comment mener à bien le notre, ainsi que quelques pistes notamment sur l'interface utilisateur.

FIGURE 1.4 – Interface de Kriegspiel par RSG



2

Besoins fonctionnels

2.1 Définition des priorités

2.1.1 Besoins primaires

L'objectif premier consiste à mettre en place une version jouable, via la console, du *Jeu de la Guerre* à l'aide d'un moteur de jeu implémentant les règles en restant le plus fidèle aux idées de Guy Debord. Ce moteur de jeu devra permettre d'affirmer si une action donnée par un joueur à un moment donné d'une partie est valide ou non. Cette implémentation devra pouvoir rendre possible l'extension du moteur pour les besoins secondaires et tertiaires.

Les besoins et contraintes de programmation apportés par cette première partie sont les suivants :

1. Mettre en place un moteur de règle respectant les règles décrites dans le *Jeu de la Guerre*.
2. Permettre à deux utilisateurs de jouer dans une même console à tour de rôle.
3. À son tour avant le début de la partie, l'utilisateur devra positionner ses unités sur le plateau sans que la position de l'armée adverse lui soit montrée.
4. Dans la phase de positionnement des unités, l'utilisateur pourra déplacer ses unités déjà placées sur le plateau sans limitation dans sa zone à l'aide de la commande permettant le déplacement d'une unité.
5. Les unités sont désignées de la façon suivante : (voir figure 2.3)
   - Le régiment d'infanterie : `I`
   - Le régiment de cavalerie : `C`
   - Le régiment d'artillerie à pied : `A`
   - Le régiment d'artillerie à cheval : `Ac`
   - L'unité de communication à pied : `R`
   - L'unité de communication à cheval : `Rc`
   - Une montagne : `M`
   - Un col : `Co`
   - Une forteresse : `F`
   - Un arsenal : `Ar`
6. À son tour, l'utilisateur devra proposer une action en utilisant la syntaxe suivante. Les actions disponibles dans un premier temps sont les suivantes (voir figure 2.3) :
   - La position de chaque case est décrite par l'utilisateur à l'aide d'une ou plusieurs lettres suivi, sans espace, d'un nombre représentant respectivement l'axe des ordonnées et des abscisses. Ainsi, la case supérieure gauche est A1 et la case inférieure droite est T25.
   - Positionner une unité au début de la partie : `set <U> <C1>` où `U` est le type de l'unité et `C1` la case où l'utilisateur veut la positionner.
   - Déplacer une unité sur une case ciblée vers une autre case : `move <C1> <C2>` où `C1` est l'unité alliée ciblée, `C2` la case d'arrivée
   - Attaquer une case en ciblant la case de l'unité attaquante : `attack <C1> <C2>` où `C1` est l'unité alliée ciblée, `C2` la case où se situe l'unité ennemie attaquée.
   - Déclarer la fin de son tour : `end`

- Déclarer forfait : `surrender`
- Proposer un match nul : `pat`
- Accepter une requête : `Y`
- Refuser une requête : `N`

7. Lorsqu'un déplacement est proposé, le programme devra lui même trouver un chemin valide pour y accéder.
8. Le programme générera et affichera un aperçu de la situation représentant l'état du plateau après chaque coup valide joué (nature de l'aperçu à définir ultérieurement, voir figure 2.2).
9. Afficher au début du prompt le numéro du joueur qui est en train de jouer.
10. Le programme ne sera pas sensible à la casse pour les entrées de l'utilisateur.

2.1.2 Besoins secondaires

Le projet se divise donc en deux parties, après avoir une version jouable du *Jeu de la Guerre* entre deux utilisateurs sur une même console, il est désormais question d'ajouter des aides aux joueurs et des interactions avec l'interface graphique.

Il s'agit d'étendre l'implémentation actuelle pour permettre de pouvoir jouer une partie complète sur le programme en utilisant des aides à la compréhension d'une situation.

Ces aides permettront à l'utilisateur d'appréhender plus facilement et rapidement la situation de la partie, à savoir par exemple, mais peut-être pas limité à, le potentiel offensif et défensif de chaque case du plateau, la portée des lignes de communications, la portée d'attaque des unités. Ces aides ne devront pas être exclusivement visuelle et pourront être utilisées par un potentiel joueur artificiel plus tard.

L'utilisateur sera en mesure de jouer toute une partie via l'interface graphique à l'aide de sa souris. Il pourra ainsi interagir avec ses unités mais aussi sélectionner les aides à afficher/enlever sur le plateau à l'aide d'un menu déroulant par exemple.

Ainsi les besoins pour cette partie sont les suivants :

1. Lorsqu'un coup est refusé par le moteur de règle, une explication du refus devra être affichée sur la console.
2. Sauvegarder l'état du plateau dans un fichier texte, en codage ASCII, en respectant un ordre logique des informations stockées, le fichier sera lisible par l'utilisateur. Cette action sera possible à l'aide de la commande : `save <File>`
3. Le fichier de sauvegarde aura le format suivant :

FIGURE 2.1 – Format de la sauvegarde

```
<Largeur>;<Hauteur>
<Joueur Actuel>; <Nombre de coups restant>
<ID d'un décor>;<X>;<Y>;<Est détruit>;<Joueur>
...
<ID de l'unité>;<X>;<Y>;<Doit bouger>;<A bougé>;<A attaqué>;<Joueur>
...
```

4. Instancier une partie à partir d'un fichier texte, en codage ASCII, respectant l'ordre logique des informations décrit par la sauvegarde. Cette action sera possible à l'aide de la commande : `load <File>`
5. Annuler la dernière action effectuée par l'utilisateur à l'aide de la commande : `revert`
6. Permettre le déplacement d'une unité qui initiera une attaque à l'issue de ce mouvement à l'aide la commande : `charge <C1> <C2> <C3>` où `C1` est l'unité alliée ciblée, `C2` la case d'arrivée de l'unité et `C3` la case où se situe l'unité ennemie attaquée.
7. Afficher/enlever les lignes de communications sur l'aperçu du plateau à l'aide de la commande : `toggleCom`
8. Afficher/enlever le potentiel d'attaque de l'utilisateur sur chaque case si il est différent de 0 à l'aide de la commande : `toggleAtk`

9. Afficher/enlever le potentiel de défense de l'utilisateur sur chaque case si il est différent de 0 à l'aide de la commande : `toggleDef`
10. Sélectionner une unité à l'aide d'un clic gauche de la souris, la case ainsi sélectionnée sera mis en évidence (surbrillance, bords plus gros, ...).
11. Sélectionner une case vide à l'aide d'un clic gauche de la souris lorsqu'une unité est sélectionnée déclenche son déplacement si celui-ci est possible.
12. Sélectionner une unité adverse lorsqu'une unité est sélectionnée déclenche une attaque si elle est possible sur la case ciblée.
13. Des boutons sur le côté du plateau ou un menu déroulant seront disponibles permettant d'afficher ou d'enlever certains modes rajoutant de l'information visuelle sur le plateau. Ces modes seront en partie constitués des précédents besoins décrits à savoir l'affichage du potentiel d'attaque et/ou de défense sur chaque case du plateau si cette valeur est différente de 0.
14. Des menus déroulant permettront d'accéder aux fonctionnalités de sauvegarde et de chargement d'un fichier comme décrit dans les besoins primaires.
15. Lors de la phase de positionnement des unités, l'utilisateur pourra choisir une unité parmi celles se trouvant dans un cadre sur l'un des côtés du plateau, pour la positionner sur la case du plateau choisie à l'aide d'un clic gauche de la souris.
16. Toujours durant cette phase, l'utilisateur pourra repositionner une unité sur le plateau sur une autre case, un échange de position se fera si la case ciblée est occupée par une unité alliée.

2.1.3 Besoins tertiaires

Dans cette partie, il s'agira de se focaliser sur un analyseur, utilisant des données telle que la cartographie des coefficients d'attaque, dont le but sera de pouvoir calculer un ou plusieurs coups à partir d'une situation donnée, en utilisant des notions stratégiques [7] pour lui faire mettre en place une tactique.

M. Narbel nous a notamment indiqué les deux besoins concernant une stratégie potentielle pour un joueur artificiel :

- Évaluation ("compréhension") d'une position (statique), (Spatial Reasoning[6] - Potential fields - PathFinding) Détermination de "points décisifs". Représentation graphique de cette évaluation (attaque, défense, communications).
- Évaluation ("compréhension") d'une suite de mouvements (dynamique). Prise en compte de techniques de planification (Planning). Représentation graphique de cette évaluation.

Pour le rendu final, au moins une stratégie et une tactique seront attendues, pouvant être utilisées en même temps, à savoir :

- Une stratégie permettant de trouver le chemin le plus efficace pour atteindre les arsenaux de l'adversaire (analyse des "vallées" ou faiblesses dans la formation adverse pour atteindre l'objectif).
- Une tactique permettant d'obtenir une configuration optimale dans la planification d'une attaque ou d'une défense (utilisation d'une cartographie des potentiels d'attaque et de défense).

D'autres stratégies pourront être mises en place si le temps le permet, à savoir mesurer l'agressivité et le positionnement des troupes de l'adversaire (réduction à une dimension pour chacun des axes), et de détecter les tactiques "commando", comme remarqué avec le détachement de l'unité de Cavalerie dans la partie présentée par Debord dans son livre (voir [5], pages 11-127).

Ces stratégies et tactiques pourront être utilisées comme représentation graphique pour conseiller le joueur (aperçus types "schémas militaires"), ainsi que comme une série d'indications pour un potentiel joueur artificiel.

FIGURE 2.2 – Ébauche d'interface commande et d'aperçu de l'état du plateau

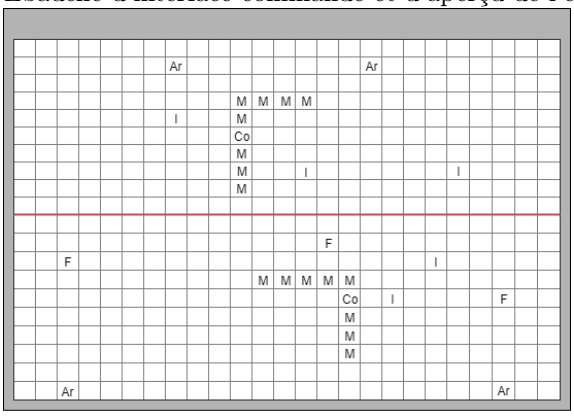
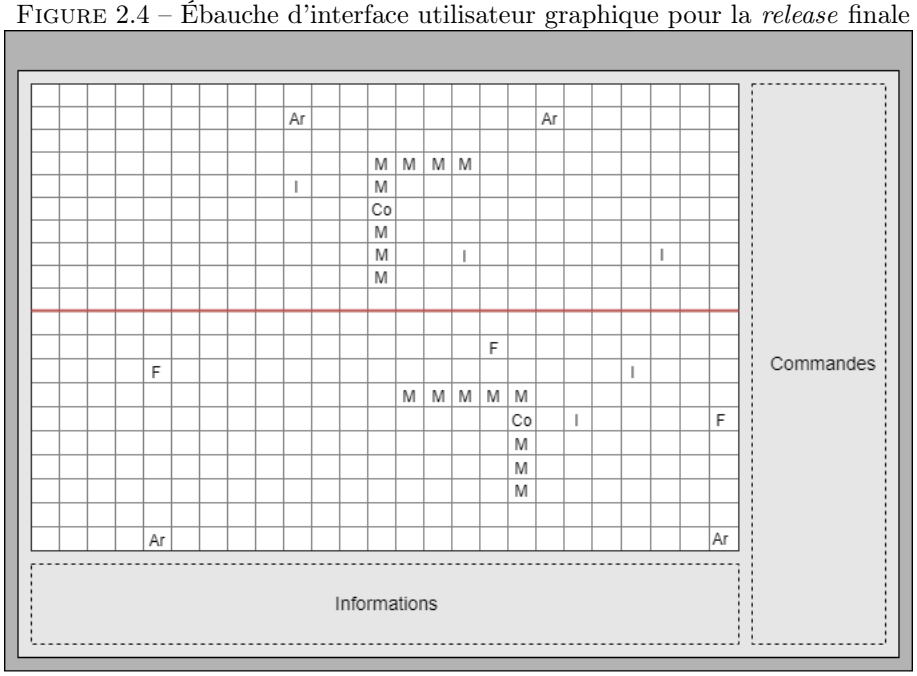


FIGURE 2.3 – Commandes et identifiants des unités et terrains employés

Commandes de jeu	
Ecriture d'une coordonée : C12, B6, A20 De A1 (coin sup. gauche) à T25 (coin inf. doit)	
Mouvement de l'unité sur la case C1 vers C2	`move <C1> <C2>`
Attaque par l'unité sur la case C1 de C2	`fight <C1> <C2>`
Charge par l'unité sur la case C1 de C2	`charge <C1> <C2>`
Afficher la liste des unités	`units`
Déclarer forfait	`surrender`
Demander egalité	`pat`
Place l'unité U sur la case C	`set <U> <C>`
Termine le tour du joueur courant	`end`
Commandes générales	
Sauvegarder la partie dans le fichier FILE	`save <FILE>`
Charger la partie depuis le fichier FILE	`load <FILE>`
Annuler les dernières commanes	`revert`

Infanterie	`I`
Cavalier	`C`
Artillerie	`A`
Artillerie montée	`Ac`
Communication	`R`
Communication monté	`Rc`
Montagne	`M`
Col	`Co`
Fort	`F`
Arsenal	`Ar`

FIGURE 2.4 – Ébauche d'interface utilisateur graphique pour la *release* finale



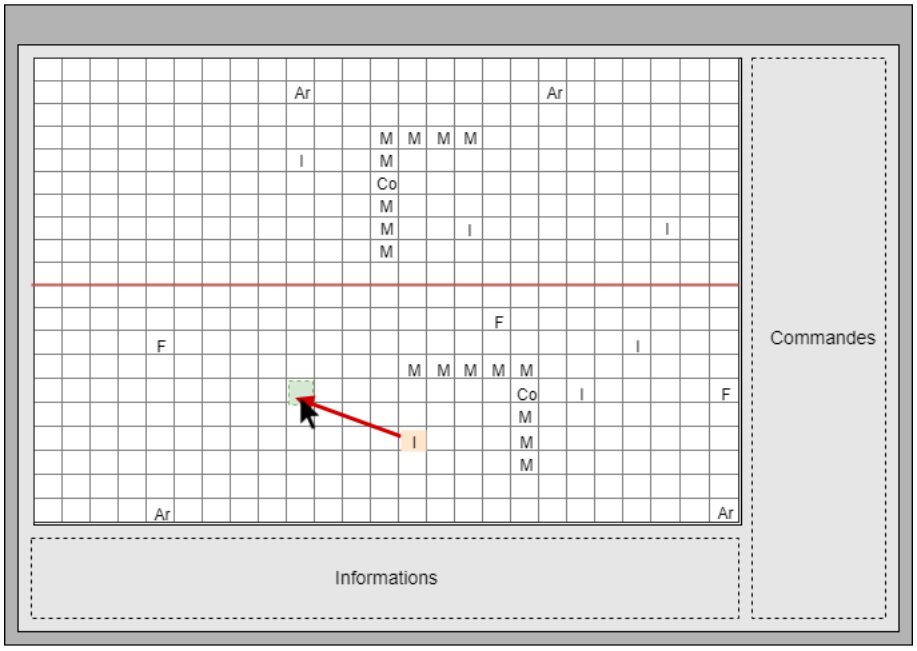
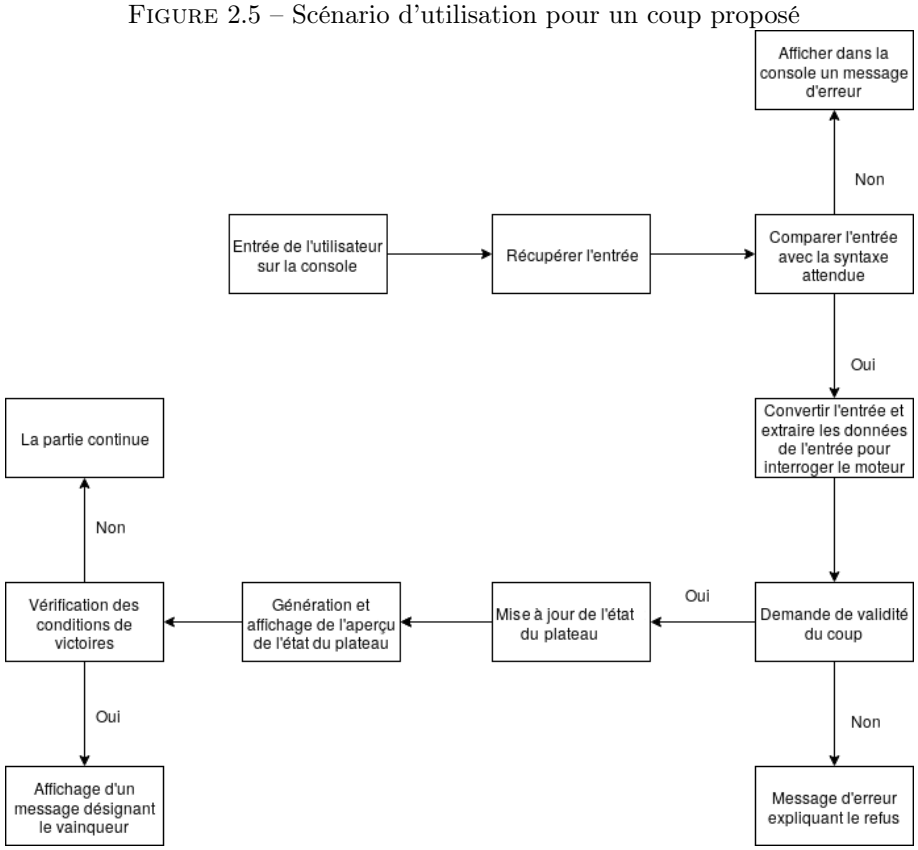


FIGURE 2.5 – Scénario d'utilisation pour un coup proposé

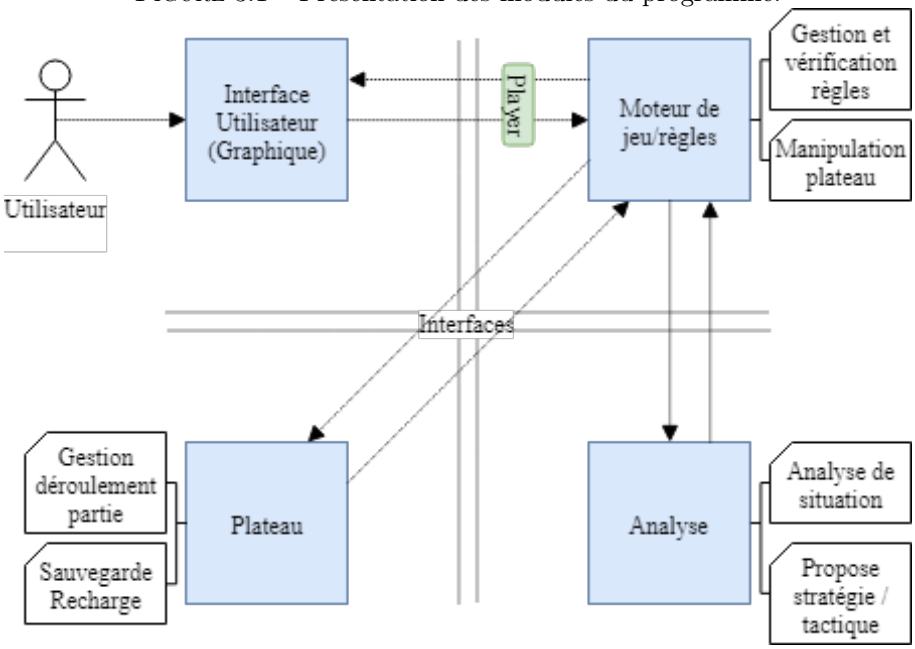


3

Besoins non-fonctionnels

1. Le code du moteur de règles devra avoir un certain degré de généricité et de modularité pour pouvoir altérer les règles du jeu lors du développement ou même après la release finale (voir figure 4 pour une idée de la modularité attendue).
2. Le temps d'attente entre un coup proposé et sa validité évaluée par le moteur de règle devra être de l'ordre de la seconde.
3. Le jeu doit fonctionner sur un système UNIX.
4. Le jeu sera codé dans le langage de programmation Java, et Scala pour les interfaces graphiques.

FIGURE 3.1 – Présentation des modules du programme.

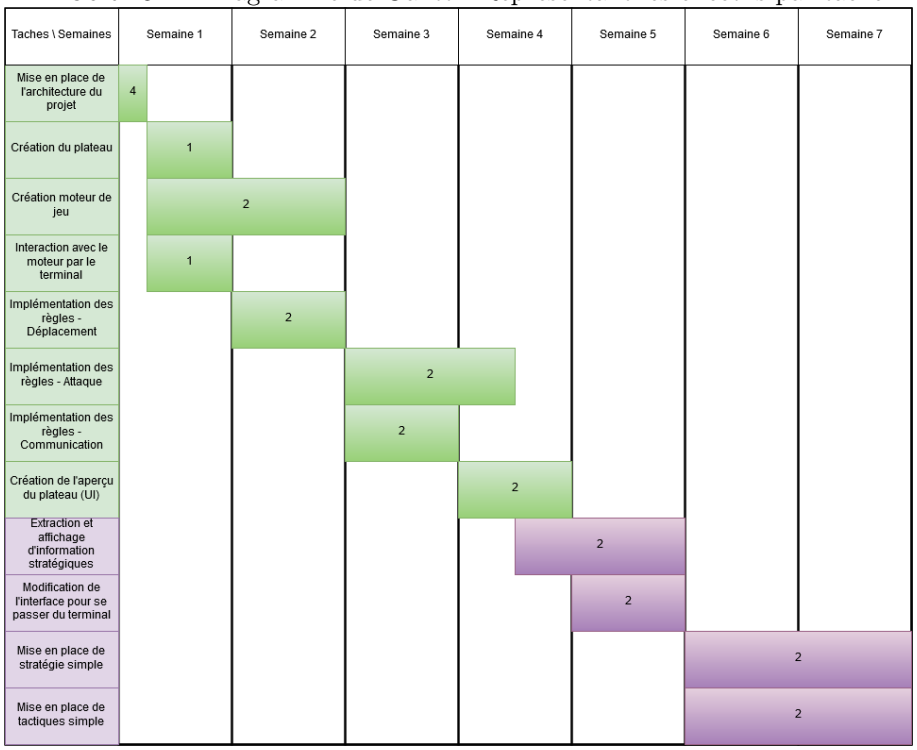


5. Le module de moteur de règles sera composé d'un ensemble de règles et s'occupe de modifier l'état du plateau sur le déroulement d'une partie.
6. Le module d'interface utilisateur offrira un environnement utilisable par l'utilisateur pour interagir avec le jeu. Les interactions avec le moteur de jeu seront définies par une interface (représentée par `Player`) qui sera aussi utilisée aussi bien que par un joueur humain que par un potentiel joueur artificiel. Cet environnement sera dans un premier temps une interface console affichant l'état du plateau dans une fenêtre à part, puis une interface graphique après la première release.
7. Le module d'Analyse permettra d'obtenir des informations et données stratégiques et tactiques sur une situation donnée et sur les tours précédents. L'idéal serait de pouvoir proposer

des stratégies ou tactiques à partir de ces informations, pouvant être ultimement employé par un joueur artificiel.

8. Le module de plateau est une sorte de conteneur qui s'occupe de gérer l'état du plateau, de l'exporter et de l'importer. Il ne doit pas être accessible par les autres classes Analyse et Interface Utilisateur pour éviter la perte de cohérence sur le déroulement de la partie. Restreindre l'accès au Plateau uniquement au moteur de jeu force les autres modules à passer par la vérification des règles avant d'agir sur le terrain.
9. Une couverture des tests sur les différents modules du jeu est attendue lors du développement. Ces tests devront être implémentés par les développeurs travaillant sur leur module au fur et à mesure.
10. Des tests en boîte noire devront être fournis, permettant de valider les interactions entre l'utilisateur et le programme. Certains de ces tests se baseront sur les configurations du plateau de la partie décrite dans le livre de Debord (voir [5], pages 14-24), notamment pour vérifier si l'état du plateau résultant du test sur l'état initial est bien conforme à ce qui est décrit. Cette partie servira aussi de témoin principal pour confirmer la validité des règles implémentées.
11. Le développement des différentes étapes non triviales devra être conduit par groupes de 2 (voir le diagramme 3.2) développeurs pour garder une cohérence, pas plus pour éviter les ralentissements sur une seule partie du code, mais pas moins non plus pour avoir un code maintenable et compris par plusieurs développeurs.
12. Les deux dernières semaines du projet seront réservées à la rédaction du mémoire ainsi que pour compenser le retard éventuel généré lors du développement.

FIGURE 3.2 – Diagramme de Gantt - Représentant les effectifs par tâche



Bibliographie

[1] Kriegspiel. http://r-s-g.org/kriegspiel/index.php. Visité : 2018-01.

[2] Bill Leeson, Johan Hörberg. *Kriegsspiel - The Prussian Army Wargame*. Too Fat Lardies, 1983. Reproduction du jeu original du XIXème siècle des père et fils von Reiswitz.

[3] Carl von Clausewitz. *Vom Kriege*. 1832.

[4] Guillaume Desbieys, David Cheminade, Hubert Mondon, Quentin Michaud. Jeu de la guerre - Kriegspiel (PdP 2014). https://services.emi.u-bordeaux.fr/projet/savane/projects/kriegspiel/. Visité : 2018-01.

[5] Guy Debord, Alice Becker-Ho. *Le Jeu de la Guerre - Relevé des positions successives de toutes les forces au cours d'une partie*. Gallimard, 2006.

[6] Michael Leece, Arnav Jhala. Reinforcement Learning for Spatial Reasoning in Strategy Games . https://www.aaai.org/ocs/index.php/AIIDE/AIIDE13/paper/view/7407/7603. Visité : 2018-02.

[7] Funge Millington. *AI for Games : chapters 5 - 6 : Decision Making - Tactical and Strategic AI*. Morgan Kaufmann Publishers, 2009.

[8] Richard H. Berg. *Desert Fox*. Six Angles, Simulations Publications Inc., 1981.