
PROJET YAHTZEE

Biard Sunny
Brouard Antoine
Champion Nicolas
Lemarchand Thibault

19 avril 2019

Table des matières

1	Introduction	3
2	Coder le jeu à un ou deux joueurs	5
2.1	Coder le premier joueur	5
2.2	Lancer de dés	6
2.3	Choix de relancer	7
2.4	Choix de placement dans la grille	7
2.5	Fin de partie	8
2.6	Coder le deuxième joueur	8
2.7	Phase de tests	8
2.7.1	Fonctions de mains.c	8
2.7.2	Fonctions de fonctions_jeu.c	9
2.7.3	Fonctions de fonctions_joueur.c	9
3	Gérer l'interface non graphique	10
3.1	Créer la matrice de grille	10
3.2	Calcul du Score	10
3.3	Résultats	11
4	Coder l'ordinateur	12
4.1	Trouver une stratégie efficace	12
4.2	Stratégie Grille supérieur	12
4.3	Stratégie Grille Inférieure	13
4.4	Application de la stratégie	14
4.5	Phase de tests	15
5	SDL (Interface graphique)	16
5.1	Initialisation de la fenêtre	16
5.2	Évènements de la fenêtre	16
5.2.1	Appui sur la souris (MOUSEBUTTONDOWN)	16
5.2.2	Relâchement de la souris (MOUSEBUTTONUP)	17

5.3	Déroulement du jeu	17
5.3.1	Version 1 joueur (joueur contre ordinateur)	18
5.3.2	Version 2 joueurs (joueur contre joueur)	18
5.3.3	Résultats	18
6	Conclusion	19
7	Annexe	20
7.1	Annexe 0 : Vue global du dépôt	20
7.2	Annexe 1 : Feuille de marque	21
7.3	Annexe 2 : Statistiques IA	22
7.4	Annexe 3 : Yahtzee - En ligne de commande	23
7.5	Annexe 4 : Yahtzee - Interface graphique	26
7.6	Annexe 5 : Exemple de débogage	27

1 Introduction

Nous travaillons sur le jeu Yahtzee qui combine hasard et logique, le jeu se joue à deux ou plus. Le but du jeu est d'obtenir plus de points que son adversaire ou ses adversaires, pour cela il faudra lancer 5 dés que l'on peut garder et ainsi valider notre jeu ou alors que l'on peut relancer deux fois maximum par manche. Il existe différentes combinaisons de dés permettant de gagner des points séparées en 2 parties :

Partie supérieure :

- As : $1 \times$ le nombre de dés 1 obtenus
- Deux : $2 \times$ le nombre de dés 2 obtenus
- Trois : $3 \times$ le nombre de dés 3 obtenus
- Quatre : $4 \times$ le nombre de dés 4 obtenus
- Cinq : $5 \times$ le nombre de dés 5 obtenus
- Six : $6 \times$ le nombre de dés 6 obtenus
- Prime : 35 points si le sous-total est supérieur ou égal à 63 points
- Total supérieur : Somme de chaque cases de la partie supérieure

Partie inférieure :

- Brelan : Il faut trois dés identiques, il rapporte la somme de tous les dés
- Petite suite : 1,2,3,4 ou 2,3,4,5 ou 3,4,5,6 elle rapporte 30 points
- Grande suite : 1,2,3,4,5 ou 2,3,4,5,6 elle rapporte 40 points
- Full : Trois dés identiques + deux dés identiques, il rapporte 25 points
- Carré : Quatre dés identiques, il rapporte la somme de tous les dés
- Yahtzee : Cinq dés identiques, il rapporte 50 points
- Chance : Elle rapporte la somme de tous les dés, à n'utiliser qu'en cas de nécessité
- Total inférieur : Somme de chaque cases de la partie supérieure
- Total : Sommes de la partie inférieure et de la partie supérieure

Cela correspond a la feuille de marque du Yahtzee (que l'on utilise lors d'une partie traditionnelle), voir annexe 1.

Pour faire ce jeu en langage C nous avons décidé de, dans un premier temps, le programmer sur la console en faisant un seul joueur, nous avons ensuite implémenté un second joueur. Deuxièmement, nous avons fait l'interface graphique avec SDL2 parallèlement de l'algorithme permettant à de faire l'intelligence artificielle. Nous n'avons malheureusement pas eu le temps de programmer la partie réseau.

Pour être le plus efficace possible dans notre projet nous avons décidé de faire un diagramme de Gantt pour les grandes lignes et nous avons fait un Trello pour nous organiser plus en détail, nous avons opté à une séparation en deux groupes (Nicolas et Thibault, Antoine et Sunny) à fin de ne pas faire tous la même chose. Chaque bout de programme a été minutieusement testé et débogué avant d'être ajouté sur le github du groupe.

Lien du dépôt github : <https://github.com/nichampion/yahtzee>

Les fichiers Gantt et Trello sont accessibles depuis le dépôt.

2 Coder le jeu à un ou deux joueurs

2.1 Coder le premier joueur

Afin de coder le joueur nous avons programmé en parallèle les fonctions de combinaisons possibles des dés du yahtzee et les fonctions de lancer, relancer et de choix de placement dans la grille.

Pour les fonctions de combinaisons, nous avons créé un fichier `mains.c` qui sert à contenir ces fonctions. En premier lieu, nous avons commencé par le plus simple, c'est à dire les combinaisons servant à remplir la grille supérieure du yahtzee.

Pour remplir cette grille, nous avons fait une fonction s'appelant `section_superieure` ayant en paramètre un joueur et un entier à tester. Par la suite, on parcourt le tableau de dés et lorsqu'un dé est égal à l'entier à tester alors on ajoute à la variable `res` la valeur du dé en question. Alors si `res` est supérieure à 0, ce qui veut dire que le tableau de dés comportait au moins un dé de la valeur de l'entier en paramètre, on renvoie `res`. Sinon on renvoie -1 pour signifier qu'il n'y a pas de dés de la valeur de l'entier en paramètre dans la main du joueur. Ensuite, pour les fonctions de la grille inférieure, nous avons commencé par les fonctions du brelan et du carré car elles sont similaire donc une fois la fonction du brelan programmée, il n'y a plus qu'à la réutiliser pour le carré en la modifiant légèrement.

Pour programmer la fonction du brelan, nous avons utilisé trois variables servant à prendre les valeurs des dés. Pour commencer, la variable `n1` prend la valeur du premier dé de la main du joueur et on incrémente le nombre de dé `n1` dans la main. Ensuite, on parcourt la main du joueur, si on trouve un autre dé de la valeur de `n1` alors on incrémente le nombre de dé `n1`, sinon si le dé suivant n'est pas égal à `n1` et que l'on a pas attribué de dé à `n2`, alors on attribue ce dé à `n2` et on incrémente le nombre de dé `n2`, ainsi si on trouve un dé égale à `n2` alors on incrémente le nombre de dé `n2`. Et on refait la même manipulation pour le dé `n3`.

A la fin pour savoir si la main du joueur dispose d'un brelan, on test les nombres de dé `n1`, `n2` et `n3` et si l'un d'entre eux est égal à 3 et les autres à un, cela veut dire que la main du joueur contient un brelan et on retourne le nombre de points que le joueur marque. Sinon on retourne -1.

Pour la fonction carré, on copie la fonction du brelan cependant on utilise seulement deux variables pour prendre la valeur des dés au lieu de trois. Ensuite, on fait les mêmes tests que dans la fonction du brelan et si nb_n1 ou nb_n2 est égale à 4 et l'autre variable à 1, cela signifie que le joueur détient un carré et on retourne le nombre de points qu'il marque sinon on retourne -1.

Pour le full, c'est presque le même cheminement que le carré cependant au lieu de tester si on détient 4 dés identiques, on regarde si on a un brelan et une paire donc si nb_n1 est égale à 2 ou 3 et que nb_n2 est égale à 3 ou 2 alors le joueur détient un full et on retourne 25 (nombre de points marqué en faisant un full). Sinon on retourne -1.

Pour les fonctions de petites et grandes suites, on crée un tableau de dés local à la fonction afin de pouvoir trier les dés dans l'ordre croissant permettant une phase de test plus simple. Ainsi si le dé actuel + 1 est égal au dé suivant de la main alors on incrémente le nombre de dés consécutifs. Sinon si le nombre de dés consécutifs n'est pas égale au nombre nécessaire pour la petite suite (c'est à dire 3) et que le dé suivant n'est pas égale au dé actuel + 1 alors le nombre de dés consécutifs est égal à 0. Ainsi si le nombre de dés consécutifs est supérieur ou égal à 3 alors le joueur détient une petite suite et on renvoie 30. Sinon on retourne -1. Pour la grande suite, c'est le même programme cependant, on n'utilise pas la variable contenant le nombre de dés consécutifs ainsi si le dés suivant n'est pas égal au dé actuel +1 alors on renvoie -1 signifiant que le joueur n'a pas de grande suite. Sinon on retourne 40.

Pour la dernière fonction testant le yahtzee, on teste juste si tous les dés de la main du joueur sont identiques. Si oui, on retourne 50, si non on retourne -1.

2.2 Lancer de dés

Pour simuler le lancement des 5 dés du joueur, nous avons créé une fonction lancer qui s'appuie sur la librairie time.h. Dans cette fonction, nous effectuons pour le dé du joueur entré en paramètre un rand entre 1 et 6 (toutes les valeurs que peuvent prendre un dé), ce qui attribue aléatoirement une valeur comprise dans cet intervalle au dé. Nous n'effectuons pas le lancer de tous les dés en une seule fois afin de faciliter les relancements de dés.

2.3 Choix de relancer

Dans le jeu du Yahtzee, le joueur a la possibilité de relancer 2 fois les dés de son choix en en mettant certains de côté. Pour ce faire, nous avons créé une fonction `relancer` qui commence par demander au joueur s'il souhaite relancer des dés. S'il répond "oui" (c'est à dire 1 dans l'invité de commandes), on lui demande pour chaque dé s'il souhaite le relancer. Si c'est le cas, on appelle la fonction `lancer` avec en paramètre le dé à relancer, ce qui lui attribuera une nouvelle valeur. Si le relancement a bien été effectué, on retourne 1, sinon on retourne 0.

2.4 Choix de placement dans la grille

Le choix de placement d'un score dans la grille est effectué par plusieurs fonctions.

Tout d'abord, la fonction `test_mains` est chargée de récupérer toutes les combinaisons de la grille que le joueur peut jouer avec sa main actuelle. Pour cela nous avons deux boucles `for` pour tester les combinaisons des sections supérieure et inférieure. La première boucle teste donc si la main du joueur permet de jouer une des combinaisons de la section supérieure (total de 1, total de 2, ...), seulement si celle-ci n'a pas déjà été jouée, par l'appel de la fonction `section_superieure` avec les dés du joueur pour chaque combinaison. Nous récupérons la valeur du retour de la fonction dans un joueur "test" afin de ne pas écraser les données du joueur. Le principe est ainsi le même pour la seconde boucle `for` qui teste les combinaisons de la section inférieure, excepté que pour celle-ci nous avons créé un pointeur de fonction qui pointe sur chacune des fonctions de la section inférieure présentes dans `main.c` (`brelan`, `carre`, `full`, `petite_suite`, `grande_suite`, `yahtzee` et `chance`). Pour finir, l'appel de la fonction `affichage_possibilites` avec en paramètres le joueur ainsi que le joueur test (qui contient les scores de chaque case de la grille que le joueur peut jouer) permet d'afficher dans l'invité de commandes la grille du jeu avec pour chaque case non remplie le score que le joueur peut obtenir en la remplissant.

Ensuite, une fois les deux fonctions précédentes exécutées, on appelle la fonction `choix_placement` qui permet au joueur de sélectionner la case qu'il veut remplir. Cette fonction se compose d'un `switch` sur la variable `choix` (qui prend une valeur comprise entre 1 et 14 entrée par le joueur) couvrant la totalité des cases de la grille du joueur. Ainsi, pour la case choisie par le joueur, si celle-ci n'a pas encore été remplie, on lui affecte le score correspondant déterminé précédemment dans `test_mains` (pour un score nul, c'est

à dire de valeur égale à -1, on affecte la valeur 0). Sinon, on affiche un message d'erreur demandant de choisir une autre case.

Enfin, le déroulement de tout ce processus de jeu est exécuté par la fonction `tour_joueur` qui va faire appel aux fonctions `lancer`, `relancer`, `test_mains`, `affichage_possibilites` (dans `test_mains`) et `choix_placement`, ce qui correspond à un tour du jeu.

2.5 Fin de partie

La fonction `fin_de_partie` détermine si toutes les cases de la grille du joueur (sauf les cases des totaux) sont remplies. Si c'est le cas, on appelle la fonction `calcul_totaux` pour obtenir le score final et on retourne 1. Sinon on retourne 0.

2.6 Coder le deuxième joueur

Pour coder le deuxième joueur, il a suffi de créer un deuxième élément `t_joueur`, de l'intégrer à notre boucle de jeu en ajoutant un appel de `tour_joueur` avec en paramètre ce deuxième joueur et enfin d'ajouter le test de fin de partie du joueur 2 à la condition if du `game_over`.

2.7 Phase de tests

2.7.1 Fonctions de mains.c

Appel des fonctions du fichier `mains.c` avec l'inclusion `"mains.h"`.

On crée un tableau de dés qui nous permettra de modifier rapidement la mains de dés, ainsi qu'un joueur `j`. Puis, on entre le tableau de dés dans le tableau attribué dans la structure `t_joueur`. Enfin, on appelle la fonction à tester en mettant la valeur renvoyée dans une variable attribué. Si la valeur renvoyée n'est pas égale à -1 alors la mains de dés possédée par le joueur détient bien la combinaison testée avec la fonction. Sinon la main du joueur ne détient pas la combinaison de dés testée.

Lors de l'appel de la fonction, pour savoir si celle-ci fonctionne bien dans tout les cas possible de la combinaison, on a testé différentes combinaisons de dés qui comportaient elles-mêmes la combinaison de la fonction. C'est à dire, pour une petite suite, nous avons d'abord placé la petite suite au début du tableau de dés (1,2,3,4,6). Une fois que nous avons vu que cette combinaison fonctionnait correctement, nous avons placé la petite

suite à la fin du tableau (6,1,2,3,4). Cela fonctionnait également, cependant, une fois après avoir placé la petite suite en mélangeant les chiffres dans le tableau, nous avons repéré que la fonction comportait un bug, c'est à dire que lorsque la fonction rencontrait un chiffre qui ne suivait pas le chiffre précédent, la fonction remettait le booléen de test à 0. Afin de remédier à ce problème, on effectue un tri dans le tableau afin de trier celui-ci par ordre croissant. Nous avons également ajouté une condition au "else if" du "for" afin de préciser que si le booléen de test était déjà à 3 (petite suite) et que le chiffre suivant n'était pas égal à $\text{tab}[i]+1$ on ne remettait pas le booléen de test à 0.

Nous avons procédé exactement de la même façon pour les autres fonctions du fichier `mains.c`.

2.7.2 Fonctions de fonctions_jeu.c

Pour tester la fonction `lancer`, on crée un joueur et on initialise ses dés à 0. Puis on appelle la fonction `lancer` pour chacun des dés du joueur. Si tous les dés ont une valeur comprise entre 1 et 6, alors la fonction `lancer` fonctionne.

Le test de la fonction `relancer` s'effectue par la création d'un joueur qui va lancer une première fois ses dés. On récupère les valeurs de chaque dé dans un tableau `"des"` puis on relance tous les dés du joueur. Enfin, on compare la valeur des dés du tableau `des` (correspondant au lancer initial) et les nouvelles valeurs des dés du joueur : si celles-ci sont différentes, alors la fonction `relancer` est opérationnelle.

Enfin, pour tester la fonction `test_mains`, on crée deux joueurs, dont un qui servira à stocker les résultats de la fonction. On lance les dés du joueur puis on appelle la fonction `test_mains` avec en paramètre les deux joueurs. Cette fonction appelle elle-même la fonction `affichage_possibilites` qui nous permet d'afficher les scores possibles. Donc, s'il n'y a aucun score possible, alors la fonction `test_mains` ne fonctionne pas.

2.7.3 Fonctions de fonctions_joueur.c

Ce module permet de gérer de manière dynamique les joueurs en mémoires. Nous avons vérifiés que la gestion s'effectue correctement grâce à l'outil Vagrind. Par ailleurs, ce module calcul les primes. Il a donc fallut vérifier qu'en fin de chaque partie, les primes apparaissaient (si cela est justifié).

3 Gérer l'interface non graphique

3.1 Créer la matrice de grille

Tout d'abord, afin de créer la matrice de grille des points du yahtzee, nous avons créé un fichier commun.h car la matrice va être utilisée dans la majorité des autres programmes.

Dans ce fichier, nous avons implémenté une structure tableau comprenant toutes les combinaisons possibles de dés après les avoir lancés (as,deux,brelan,yahtzee,etc). Toutes les variables de cette structure sont des variables d'entiers (int) car elles vont contenir les scores renvoyés par les fonctions de calcul du score dans la mains de dés.

Par la suite, nous avons créé une autre structure joueur qui contient la grille de score, la combinaison de dés qu'il a en main et un tableau de caractères contenant le nom du joueur.

Afin de faciliter le codage et l'affichage, nous avons créé des variables globales afin de définir un nombre de dés et la taille du nom du joueur.

Dans un autre fichier s'appelant fonctions_joueur.c, nous avons fait une fonction init_bloc_note ayant en paramètre une structure joueur. Le but de cette fonction est d'initialiser les cases de grille du score à -1 afin de dire que les cases n'ont pas été remplies.

Ensuite, la fonction creer_joueur ayant en paramètre un tableau de caractère contenant le nom du joueur nous sert à créer un joueur en initialisant sa structure.

Nous avons également fait une fonction detruire_joueur prenant en paramètre un double pointeur sur un joueur servant à détruire un joueur.

3.2 Calcul du Score

Le calcul du score s'effectue dans plusieurs fonctions. Dans le fichier fonctions_joueur, la fonction prime_yahtzee permet d'ajouter 100 points supplémentaires au joueur si celui-ci fait un yahtzee alors qu'il en avait déjà fait un auparavant. La fonction calcul_totaux calcule le total des points en fin de partie.

3.3 Résultats

Voir en annexe 3 les captures d'écrans. La première montre l'interface qui se présente à un humain pour le relancement de sa main. La seconde l'écran de fin de partie entre un humain et l'ordinateur.

4 Coder l'ordinateur

4.1 Trouver une stratégie efficace

Afin de trouver une stratégie efficace, nous avons effectué des recherches pour voir s'il existait une stratégie viable pour le yahtzee. Après avoir trouvé plusieurs stratégies, nous avons construit notre propre stratégie malgré le fait qu'il n'existe aucune stratégie optimale du fait du caractère aléatoire du yahtzee.

Notre stratégie consiste donc à remplir en priorité la grille supérieure de la feuille (c'est à dire les as, deux ,trois, etc, jusqu'à six) pour obtenir le bonus de 35 points. Pour cela, il faut avoir au moins 3 dès de chaque valeur pour obtenir le bonus. Par ailleurs, pour obtenir ce bonus, il faut savoir sacrifier un yahtzee. Ainsi, si on obtient un yahtzee de quatres, de cinq ou de six et que l'on n'a pas rempli la case de la grille supérieure correspondante, on sacrifie le yahtzee car $6 \times 5 = 30$ plus le bonus de 35 pts = 65 pts donc cela est supérieur aux points rapportés par un yahtzee (50 pts). Dans un deuxième temps, la stratégie consiste à avoir le plus rapidement possible les suites (grande et petite). Enfin, on se focalise sur l'obtention des autres mains (Carré, Breelan, Full, ...).

4.2 Stratégie Grille supérieur

Afin de coder la stratégie, on crée un tableau d'entier contenant le nombre de dés correspondant pour une valeur donnée dans la main du joueur.

Tout d'abord, on test le joueur pour savoir si celui-ci n'a pas rempli l'entièreté de la grille supérieure. Si c'est le cas, on n'applique pas la stratégie car celle-ci à été appliqué précédemment pour remplir la grille supérieure .

Ensuite, on parcourt le tableau crée précédemment et on le remplit avec la fonction `compter_des`. La fonction `compter_des` calcule le nombre de dés correspondant au paramètre `nb_test` (la valeur à compter) dans la main du joueur.

Par la suite, on crée une variable `val_des_a_garder` dans laquelle on va mettre le résultat de la fonction `choix_des_strat_sup`. Cette fonction nous permet de choisir les dés à garder pour marquer le maximum de score dans la partie supérieure.

Ensuite, on relance les dés qui ne sont pas égaux à cette valeur afin d'obtenir plus de dés égaux à celle-ci. Nous avons fait une boucle `for` afin de définir le nombre de relancer à effectuer (c'est à dire 2). Ainsi si le dé `i` n'est pas égal à la valeur du dé à garder alors on relance ce dé. Pour finir, on affiche la main du joueur. Si jamais on a obtenu un yahtzee, on appelle la fonction `utiliser_yahtzee`, qui détermine si on peut l'utiliser. Ensuite, on est

dans le cas où l'on ne joue pas de yahtzee. On regarde si cela est pertinent d'appliquer cette stratégie : On doit avoir au moins 3 dès identiques. Si oui, on joue cette main. Sinon, une autre fonction va déterminer la case à jouer qui rapporte le plus de points.

4.3 Stratégie Grille Inférieure

Premièrement, on appelle la fonction `strat_p_g_suite` qui permet d'appliquer une méthode permettant d'obtenir une suite (petite ou grande). Si au moins une des 2 cases correspondantes dans la feuille de marque du joueur passé en paramètre est disponible on applique cette stratégie. Tout d'abord on compte les dès et on analyse la main. On choisit une séquence du style (2-3-4, 3-4-5, ...) en fonction de la main du joueur pour essayer de la compléter. Cette séquence correspond aux valeurs à garder. Ensuite on relance, tant que l'on a pas une grande ou petite suite, les dès qui ne sont pas égaux aux valeurs à garder. Enfin, si on a une petite ou grande suite et qu'on peut l'utiliser, on le fait.

Pour la stratégie de brelan, on regarde si le joueur n'a pas déjà effectué de brelan plus tôt dans la partie, si c'est le cas, alors on n'applique pas la stratégie. Tout d'abord, on compte le nombre de dés dans la main du joueur en fonction du chiffre du dé (par exemple : 2 dés de 5). Ainsi cela va nous permettre de garder la valeur du dé étant présente en plus grand nombre dans la main du joueur. Par la suite, on relance tant qu'on dispose encore de relancer. Si le dé dans la main du joueur n'est pas égale à la valeur précédemment défini alors on relance ce dé. Une fois la phase de relance terminée, on teste la main du joueur ce qui va permettre l'affichage avec les possibilités disponibles pour le joueur. Cependant dès que l'ordinateur dispose d'un brelan, alors on valide ce brelan et on quitte le programme de la stratégie. Si aucune stratégie de brelan n'a pu être appliquée alors on retourne 0.

Pour la stratégie du carré, elle est identique à celle du carré sauf qu'au lieu de tester le brelan, on teste si l'ordinateur dispose d'un carré et si c'est le cas, alors on quitte le programme de la stratégie.

Cependant, la stratégie du full est plus complexe car elle nécessite plus de tests. En effet, le début du programme est identique aux deux autres programmes précédents mais une fois arrivé aux phases de relance, le programme se complexifie. Dans ce programme, il y a trois cas à tester. Le premier cas est le cas où l'ordinateur

dispose de quatre dés identiques. Pour répondre à ce problème, la valeur du dés à garder sera de la valeur des quatre dés. Ainsi lors de la relance, on relancera un seul dé pour essayer que celui-ci soit égal au cinquième dé de la main de l'ordinateur.

Le deuxième cas est d'avoir trois dés identiques sans avoir également de paire. Si c'est le cas alors on relance un des deux dé qui n'est pas égal à la valeur des trois dés identiques pour essayer d'obtenir une paire.

Le troisième cas est le plus complexe car il existe plusieurs sous cas. En effet dans le cas d'avoir moins de 3 dés identiques, il est possible d'avoir deux paires mais également d'avoir une seule paire et même aucune paire en mains.

Pour répondre au cas des deux paires en mains, on relance entièrement le dé qui n'est dans aucune des deux paires pour essayer d'obtenir un brelan et ainsi valider le full.

Pour le cas où l'ordinateur ne dispose que d'une seule paire, on relance tout les dés qui ne sont pas égaux à la valeur de la paire.

Pour le cas où l'ordinateur ne dispose d'aucune paire, on relance tout les dés.

A la fin de la phase de relance, on teste si l'ordinateur dispose d'un full et si c'est le cas on sort de la stratégie.

4.4 Application de la stratégie

Pour appliquer la stratégie, il est nécessaire d'appeler les fonctions dans un certains ordre.

En effet, dans un premier temps, on teste si l'ordinateur ne dispose pas d'un yahtzee, si c'est le cas alors on appelle la fonction `utiliser_yahtzee` afin de regarder si on sacrifie le yahtzee ou non.

Dans un deuxième temps, on applique la stratégie liée à la grille supérieure puis la stratégie liée aux suites.

Dans un troisième temps, on appelle les stratégies des brelan, carré et full.

Dans un quatrième temps, on appelle la stratégie liée au yahtzee.

Ces fonctions prennent toutes en paramètres un pointeur vers un compteur indiquant le nombre de relances disponible. Cela est utile, si une stratégie ne peut être appliquée et qu'on enclenche une stratégie, il ne faut pas relancer plus de 2 fois.

Pour finir, on appelle la fonction `meilleur_score` si aucune stratégie n'a pu être appliqué auparavant.

Chaque stratégie vérifie que la ligne correspondante dans la feuille de marque est dispo-

nible. Avant chaque appel de stratégie, on doit impérativement vérifier que la main qu'on passe en paramètre n'est pas déjà un brelan, carré, full, yahtzee, etc

4.5 Phase de tests

Afin de tester la stratégie pour remplir la grille supérieure du yahtzee, nous avons désactivé le lancement automatique des dés en appelant la fonction lancer et on crée un tableau d'entier que l'on appelle `dés_test` dans lequel on met la main de dés nécessaire au test. Nous avons ensuite copié le main de l'interface non graphique en faisant jouer seulement l'ordinateur. Pour tester si la stratégie précédemment codée est bien viable et efficace, nous avons simulé 10000 lancers et écrit le résultat dans un fichier .csv afin d'étudier la tendance du score de l'IA ainsi que le minimum du score possible, le maximum du score possible mais également la moyenne et la médiane des différentes parties.

Ici, on peut voir que le score maximal de l'IA est de 311 donc assez proche du score maximal possible au yahtzee (390) donc nous pouvons dire que la stratégie est viable et assez efficace. (Cf en Annexe 1).

5 SDL (Interface graphique)

5.1 Initialisation de la fenêtre

Pour commencer, il faut définir toutes les variables dont nous avons besoin pour notre fenêtre : `SDL_Window` (la fenêtre), `SDL_Surface` (surface d'un élément), `SDL_Texture` (texture à afficher), `SDL_Renderer` (rendu), `SDL_Rect` (rectangle de dimensions x,y,w,h) et `TTF_Font` (police d'écriture). Pour afficher la grille de score et le bouton lancer par exemple, nous avons créé des textures à partir d'une surface et d'une image que l'on importe. Une fois la texture créée, il n'y a plus qu'à récupérer ses dimensions (`SDL_QueryTexture`) et à l'ajouter au rendu (`SDL_RenderCopy`). Le rendu se fait par `SDL_RenderPresent(renderer)`. Le procédé est le même pour tous les autres éléments (dés, score, messages...).

L'affichage de la fenêtre ainsi que de ces éléments de base se fait dans le case `WINEVENT_SHOWN` du switch sur les événements. Cette fenêtre s'actualise en continu grâce à deux boucles while imbriquées et celle-ci se ferme uniquement si l'on clique sur la croix de la fenêtre pour la fermer (case `SDL_Quit`).

5.2 Évènements de la fenêtre

Pour réaliser le jeu nous avons besoins de créer des événements dans la fenêtre que le joueur peut utiliser pour jouer. Nous nous sommes servis de deux types d'évènements : les événements d'appui et de relâchement de la souris.

5.2.1 Appui sur la souris (MOUSEBUTTONDOWN)

Les événements de clics correspondent à des positions dans la fenêtre où le joueur peut cliquer, ce qui aura pour effet d'exécuter différentes tâches.

Tout d'abord, nous avons créé un événement de clic sur le bouton "lancer" (limité à 3 clics par tour soit 3 lancers). Un clic sur cet événement va donc lancer tous les dés du joueur (dans le cas d'un premier lancer) ou en relancer certains (deuxième ou troisième lancer), puis les afficher. L'affichage de chaque dé se fait par deux switch : le premier sur le dé du joueur, afin de déterminer la valeur de celui-ci (on stocke cette valeur dans "decourant"). Le deuxième switch se fait sur le numéro du dé en question (chaque case du switch correspondant à 1 des 5 dés et ayant chacun leurs propres coordonnées dans la

fenêtre), qui va donc afficher telle valeur de dé à telles coordonnées et initialiser dans le tableau "de_pos" le dé en position 0 (c'est à dire qu'il n'est pas mis de côté). Après avoir lancé les dés, nous appelons la fonction test_mains pour déterminer les combinaisons possibles de la main du joueur, et nous les affichons en rouge dans la grille du joueur, et s'il n'y a pas de combinaison possible pour une case, elle reste vide (si une case est déjà remplie, alors nous n'effectuons aucune action sur celle-ci).

Ensuite, les événements de clics sur les dés permettent au joueur d'en mettre un ou plusieurs de côté afin de relancer uniquement ceux sur lesquels il n'a pas cliqué (et inversement pour ceux mis de côté). Pour ce faire, nous avons un pointeur sur `SDL_Texture` qui pointe sur l'image correspondant au dé en question. On commence par effacer le dé de la fenêtre, puis on l'affiche à son nouvel emplacement grâce au pointeur et on met à jour la position de ce dé dans "de_pos" (1 s'il est mis de côté, 0 sinon).

Enfin, nous avons ajouté un dernier événement qui permet de rendre cliquable toutes les cases non remplies de la grille du joueur. Ainsi, le joueur peut choisir la case qu'il veut remplir en cliquant dessus. Le potentiel score que nous avons affiché en rouge est alors affiché en noir (on affiche 0 si la case était vide) dans cette case, on met à jour la structure du joueur et on réinitialise le compteur de lancer.

5.2.2 Relâchement de la souris (MOUSEBUTTONUP)

L'évènement de type `MOUSEBUTTONUP` nous permet d'automatiser la fin de partie. En effet, à chaque tour de jeu, on teste si `fin_de_partie` est vraie pour tous les joueurs, et si tel est le cas on affiche tous les totaux (totaux supérieur, inférieur, global et bonus) dans les cases correspondantes, on nettoie la partie de l'écran qui servait à lancer les dés (les événements d'appui ne sont évidemment plus exécutables) et on y affiche un message indiquant le joueur gagnant et son score.

5.3 Déroulement du jeu

Afin de pouvoir jouer au jeu à 1 ou 2 joueurs, nous avons créé un pointeur sur la structure `t_joueur`, afin de pointer sur le joueur courant pour chaque tour de jeu.

5.3.1 Version 1 joueur (joueur contre ordinateur)

Dans le cas de la version 1 joueur contre l'ordinateur, nous avons ajouté un nouvel évènement de type `MOUSEBUTTONUP` qui simule automatiquement le tour de l'ordinateur si c'est à son tour de jouer (on vérifie la valeur du pointeur sur le joueur courant). Si c'est le cas on appelle la fonction `tour_ordinateur` pour déterminer la combinaison qu'il va jouer. Puis, on met à jour la grille de l'ordinateur avec le score qu'il vient de faire. Enfin on affiche de nouveau le bouton "lancer" pour le joueur et celui-ci devient le nouveau joueur courant.

Pour le tour du joueur, il se sert des évènements d'appui sur la souris explicités précédemment afin de jouer.

5.3.2 Version 2 joueurs (joueur contre joueur)

Dans le cas de la version 2 joueurs, on change simplement la valeur du pointeur sur le joueur courant après chaque coup joué pour passer au tour suivant (évènement d'appui sur une des cases non remplie du joueur courant). Également, puisque les grilles des deux joueurs ont des coordonnées différentes dans la fenêtre, nous avons ajouté des conditions `if` sur le pointeur afin d'affecter les bonnes coordonnées pour l'affichage des score et le remplissage des cases.

5.3.3 Résultats

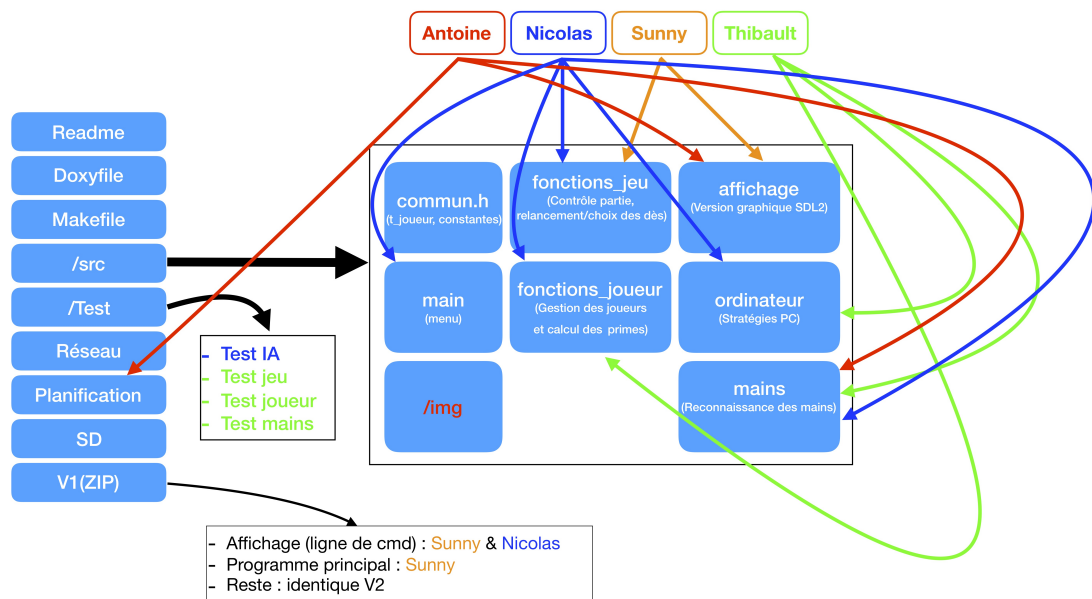
Voir en annexe 4 la capture d'écran montrant l'écran de fin de partie entre 1 humain et l'ordinateur et la capture montrant une partie en cours (Le joueur est en train de se constituer une main).

6 Conclusion

Le jeu fonctionne bien, la stratégie est assez performante. Cependant le planning à été difficilement respectable. Ainsi du retard à été pris et la mise en place d'un réseau fut impossible. Ce projet nous a permis de mettre en oeuvre nos compétences en programmation. Nous avons appris à travailler efficacement avec l'outil github. Nous avons appris à travailler ensemble dans le cadre d'un projet commun avec une échéance. En améliorations, il faudrait faire un menu graphique, encore améliorer les performances de la stratégie (invincibles).

7 Annexe

7.1 Annexe 0 : Vue global du dépôt



7.2 Annexe 1 : Feuille de marque

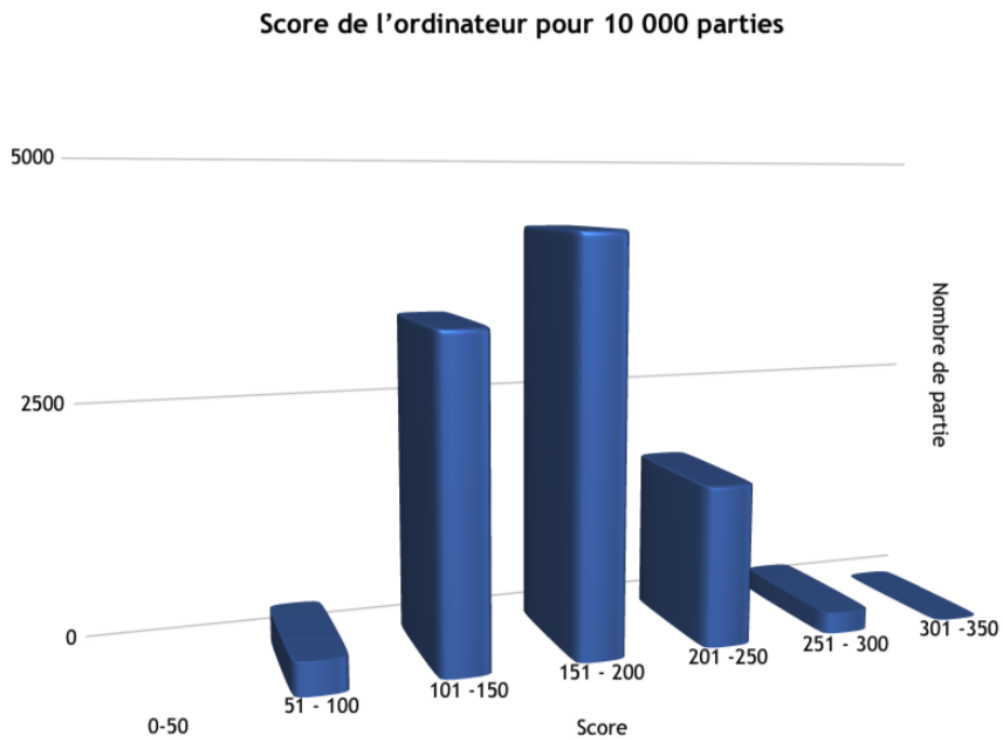
UN <i>Total des As</i>				
DEUX <i>Total des Deux</i>				
TROIS <i>Total des Trois</i>				
QUATRE <i>Total des Quatre</i>				
CINQ <i>Total des Cinq</i>				
SIX <i>Total des Six</i>				
Sous-Total de la Partie Sup.	<input style="width: 40px;" type="text"/>	<input style="width: 40px;" type="text"/>	<input style="width: 40px;" type="text"/>	
BONUS de 35 points <i>(Si le sous-total dépasse 62)</i>				
Total de la Partie Supérieure	<input style="width: 40px;" type="text"/>	<input style="width: 40px;" type="text"/>	<input style="width: 40px;" type="text"/>	

BRELAN <i>(3 dés identiques) - Total des dés</i>				
CARRÉ <i>(4 dés identiques) - Total des dés</i>				
FULL <i>(Brelan et une Paire) - 25 points</i>				
PETITE SUITE <i>(Suite de 4 dés) - 30 points</i>				
GRANDE SUITE <i>(Suite de 5 dés) - 40 points</i>				
(5 dés identiques) <i>50 points</i>				
CHANCE <i>Total des dés</i>				
Total de la Partie Inférieure	<input style="width: 40px;" type="text"/>	<input style="width: 40px;" type="text"/>	<input style="width: 40px;" type="text"/>	

Résultats

YAHTZEE™ est une marque déposée et appartient à HASBRO®.
 Cette grille a été créée par Jouetopia.fr.

7.3 Annexe 2 : Statistiques IA



Statistiques

Minimun	57
Maximun	311
Moyenne	165,644564456446
Médiane	165
Premier quartile	134
Troisième quartile	192

7.4 Annexe 3 : Yahtzee - En ligne de commande







```

* * * * * TOUR H * * * * *
*
*
* -----
* |                                     |
* |                                     |
* |-----|
* | 1 As          0 |
* | 2 Deux        0 |
* | 3 Trois       -1 |
* | 4 Quatres     4 |
* | 5 Cinq        5 |
* | 6 Six         12 |
* | Total        -1 |
* | Prime        -1 |
* |-----|
* | 7 Brellan     26 |
* | 8 Carre       -1 |
* | 9 Full        -1 |
* | 10 P Suite    -1 |
* | 11 G Suite    -1 |
* | 12 Yahtzee    -1 |
* | 13 Chance     21 |
* | Total        -1 |
* |-----|
* | TOTAL         -1 |
* |-----|
*
*
* -----
* | 1      2      3      4      5 |
* | 4      1      1      6      1 |
* |-----|
*
* * * * *
Voulez vous relancer ? (0 : Non, 1 : Oui) 1
Voulez vous relancer le dé n°1 ? (0 : Non, 1 : Oui) 1
Voulez vous relancer le dé n°2 ? (0 : Non, 1 : Oui) 1
Voulez vous relancer le dé n°3 ? (0 : Non, 1 : Oui) 1
Voulez vous relancer le dé n°4 ? (0 : Non, 1 : Oui) 1
Voulez vous relancer le dé n°5 ? (0 : Non, 1 : Oui) 1







```


YAHTZEE		
	H	PC
As	0	3
Deux	0	6
Trois	3	9
Quatre	4	12
Cinq	5	15
Six	12	18
Total	24	98
Prime	0	35
Brelan	26	7
Carre	0	7
Full	25	0
Petite Suite	0	30
Grande Suite	0	0
Chance	21	21
Yahtzee	0	0
Total	72	65
TOTAL	96	163

7.5 Annexe 4 : Yahtzee - Interface graphique


	J1	J2
1 [total de 1]  = 1	2	4
2 [total de 2]  = 2	6	4
3 [total de 3]  = 3	9	12
4 [total de 4]  = 4	4	16
5 [total de 5]  = 5	15	15
6 [total de 6]  = 6	12	24
Bonus si > à 62 [35]	0	35
Total supérieur	48	110
Brelan (3 id.) [total]	22	23
Carré (4 id.) [total]	26	18
Full House [25]	0	25
Petite suite [30]	30	30
Grande suite [40]	40	0
5 identiques [50]	0	0
Chance [total]	18	23
Total inférieur	136	119
Total	184	229

Le joueur Ordinateur a gagné avec 229 points !


	J1	J2
1 [total de 1]  = 1	1	
2 [total de 2]  = 2	6	8
3 [total de 3]  = 3	3	
4 [total de 4]  = 4	4	
5 [total de 5]  = 5	5	
6 [total de 6]  = 6		
Bonus si > à 62 [35]		
Total supérieur		
Brelan (3 id.) [total]		
Carré (4 id.) [total]		
Full House [25]		
Petite suite [30]	30	
Grande suite [40]	40	
5 identiques [50]		
Chance [total]	15	
Total inférieur		
Total		

J1

Veuillez choisir une case a remplir ou selectionner les des a garder



LANCER



7.6 Annexe 5 : Exemple de débogage

Voir le fichier PDF se situant à la racine du dépôt git.

Lien vers le fichier :

https://github.com/nichampion/yahtzee/blob/master/Exemple_Debug_1.pdf

Nom du fichier : Exemple_Debug_1.pdf

Description : Exemple de débogage d'un problème pour les relances des dès de l'ordinateur
(Lors de la stratégie)