ESEO Paris
Cycle ingénieur $\mathfrak{E}_{3a}$
2022-2023 Semestre 5

Projet : Jeu de Morpion
Cahier des spécifications techniques

Table des matières

1 Introduction **2**

2 Conception **2**
2.1 Architecture . . . 2
2.2 Interaction entre les modules . . . 3
2.3 Règles de développement . . . 3
2.3.1 Nommage des modules et fonctions . . . 3
2.3.2 Documentation et Doxygen . . . 4
2.3.3 Gestion de configuration et compilation conditionnelle . . . 4

3 Étape 0 **5**
3.1 Fonction isGameFinished . . . 5

4 Étape 1 **5**
4.1 Fonction isGameFinished . . . 5
4.2 Fonction Board_putPiece . . . 5
4.3 Fonctions Board_init, Board_free et Board_getSquareContent . . . 5
4.4 Module *board_view_text* . . . 5
4.5 Module *player_manager_mock* . . . 6
4.6 Module *Game* . . . 6
4.7 Récapitulatif des actions à réaliser . . . 7

5 Etape 2 **7**
5.1 Fonction PlayerManager_oneTurn . . . 7
5.2 Validation . . . 7
5.3 Récapitulatif des actions à réaliser . . . 8

6 Etape 3 **8**
6.1 Fonction BoardView_displayAll . . . 8
6.2 Fonction BoardView_displaySquare . . . 8
6.3 Fermeture de la fenêtre . . . 9
6.4 Prise en compte du clic souris . . . 9
6.5 Fonction tryMove . . . 10
6.6 Validation . . . 10
6.7 Récapitulatif des actions à réaliser . . . 11

7 Aller plus loin. . . 11

1 Introduction

Le tic-tac-toe, aussi appelé ”morpion” et ”oxo” en Belgique, est un jeu de réflexion se pratiquant à deux joueurs, tour par tour, dont le but est de créer le premier un alignement [source wikipedia].

2 Conception

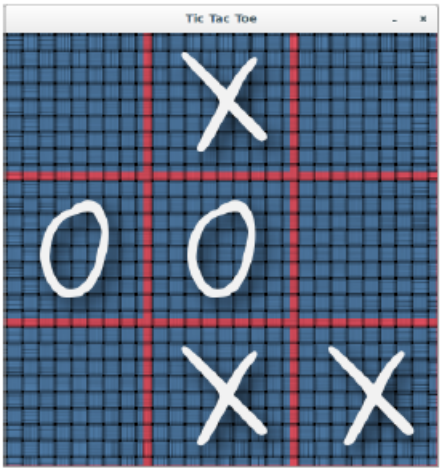

FIGURE 1 – IHM Proposée pour le jeu de Morpion à réaliser

Ce document est un guide de réalisation d'un jeu de Morpion (figure 1). L'objectif est ici de vous donner quelques ”recettes” de fabrication d'un tel jeu pour vous amener à faire le votre. Ainsi, certaines parties (les plus triviales) vous sont données afin que vous puissiez vous concentrer sur les parties les plus intéressantes de point de vue pédagogique.

2.1 Architecture

La figure 2 présente l'architecture de la solution proposée pour la conception du jeu, structurée en plusieurs modules.

Les modules *game* et *board* constituent la base logique du jeu (essentiellement les règles du jeu), réutilisable quelle que soit l'interface développée. Dans un premier temps, c'est sur cet aspect du jeu que nous allons concentrer nos efforts. Comme il est d'usage en développement logiciel, nous développons d'abord un moteur solide avant d'élaborer l'interface homme-machine.

Le module *board_view* comprend les fonctions nécessaires à l'affichage à destination de l'utilisateur.

Le module *player_manager* est l'interprète des actions de l'utilisateur, traduisant ainsi les modalités techniques (clic souris, saisie clavier) en action logique sur le jeu.

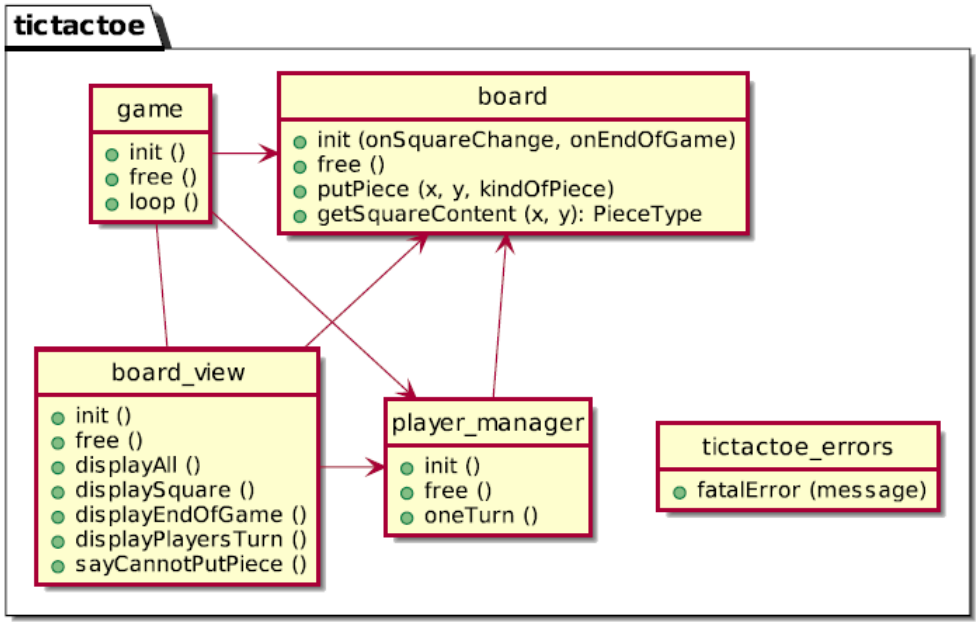

FIGURE 2 – Architecture du jeu de Morpion

2.2 Interaction entre les modules

Le diagramme de séquence de la figure 3 présente une situation où un joueur joue un coup et que la partie se termine, en faisant abstraction des paramètres effectivement échangés et du résultat de la partie. L'accent est mis sur le rôle de chaque module vis-à-vis des autres.

Dans cet exemple, la demande d'un coup émane du module *Game* en appelant la fonction oneTurn de *PlayerManager* qui capture l'action de l'utilisateur et l'envoie au module *Board*. Ce dernier appelle successivement les fonctions SquareChangeCallback et EndOfGameCallback du module *Game* et dont les adresses ont été renseignées à l'initialisation du module *Board*. Le module *Game* est donc en charge de coordonner l'ensemble du jeu. Ces fonctions font à leur tour appel à des fonctions du module d'affichage *BoardView*.

2.3 Règles de développement

2.3.1 Nommage des modules et fonctions

Pour traduire la conception résumée ci-dessus, nous appliquons un nombre de règles de nommage :

1. Les fichiers portent les noms des modules correspondants : *board_view.h* et *board_view.c* pour le module *board_view* ;
2. Les fonctions ont pour préfixe le nom du module : PlayerManager_oneTurn pour la fonction oneTurn du module *player_manager*.

Notez l'utilisation de la notation dite "CamelCase" dans les noms des fonctions (avec le caractère "underscore" pour séparer le nom du module du nom de la fonction). Pour les fichiers, nous avons

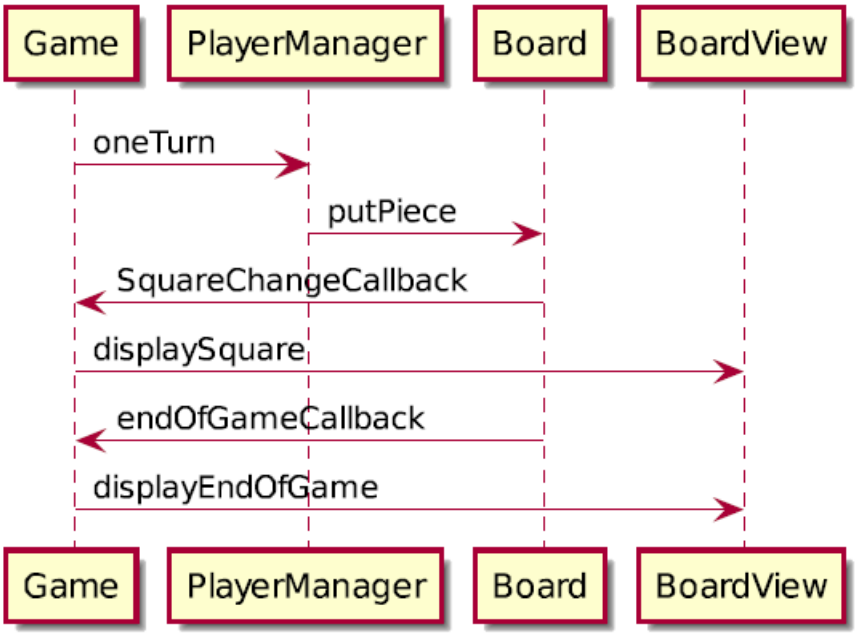

FIGURE 3 – Séquence d'un coup provoquant la fin de partie

retenu la notation exclusivement en minuscule avec "underscores" pour séparer les mots [1].

2.3.2 Documentation et Doxygen

La documentation externe des modules, types, constantes et fonctions publics est disponible dans les fichiers d'en-tête respectifs (".h").

Quoi de plus naturel, pour coder une fonction, que de connaître les modalités précises du service à rendre ? Il est donc indispensable que vous preniez connaissance de la documentation des fonctions que vous allez réaliser (les indications du présent document ne sont qu'un complément destiné à vous orienter dans les actions de réalisation).

Si vous souhaitez une lecture plus confortable de cette documentation, vous pouvez extraire la documentation Doxygen en vous basant sur le fichier de configuration (*Doxyfile*) fourni avec le code source sur le campus.

2.3.3 Gestion de configuration et compilation conditionnelle

Dans la suite de ce document, vous aurez à remplacer une implémentation d'un module par une autre (typiquement pour *board_view*). Dans ce cas, nous optons pour que le choix de la configuration (c'est à dire la sélection de l'implémentation pour chaque module) se fasse au moment de la construction.

En langage C, l'utilisation des directives de précompilation telles que "#ifdef *macro*" permet la configuration dynamique. Le code source fourni sur le campus est déjà préparé à cet effet ; il suffit ensuite de paramétrer correctement la construction.

1. Sous UNIX, l'usage veut que les noms des fichiers soient, sauf rares exceptions, écrits uniquement en minuscule

3 Étape 0

Dans un premier temps, vous allez vous concentrer sur le développement de la logique du jeu, en faisant abstraction de l'interface. Dans l'étape 0, nous nous intéressons au module *board*.

Vous trouverez le code source de base des étapes sur le campus.

3.1 Fonction isGameFinished

Écrivez le corps de cette fonction en vous aidant de la documentation Doxygen de la fonction (dans *board.c*). Le code source d'une suite de tests unitaires de cette fonction est fourni pour cette étape. Vous pouvez l'utiliser et également l'étendre pour procéder à la validation de cette fonction.

4 Étape 1

Cette étape constitue le démarrage réel de votre projet. Vous allez commencer par réaliser le moteur du jeu, donc principalement les modules *board* et *game* en vous aidant des modules *board_view_text* et *player_manger_mock* pour donner une interface textuelle simple du jeu.

Une nouvelle version du module *board* est donnée dans cette étape. Les entêtes ".h" des modules ne doivent en aucun cas être modifiés. Vous avez la possibilité d'ajouter les variables globales internes aux modules qui sont nécessaires pour le développement de la solution.

4.1 Fonction isGameFinished

Cette fonction est maintenant privée, donc interne au module, mais son fonctionnement reste celui validé dans l'étape 0.

4.2 Fonction Board_putPiece

Cette fonction doit s'assurer que la case demandée est bien vide (voir documentation Doxygen).

Lorsque le pion spécifié (croix ou cercle) est placé, il faut vérifier si la partie est terminée en utilisant isGameFinished.

Il faudra également appeler les fonctions SquareChangeCallback et EndOfGameCallback (adresses fournies lors de l'appel à Board_init) aux moments adéquats. Ces fonctions vont se charger ensuite du lien avec l'interface (figure 3).

4.3 Fonctions Board_init, Board_free et Board_getSquareContent

La documentation de ces fonctions devrait être suffisante pour vous permettre de compléter le corps de ces fonctions.

4.4 Module *board_view_text*

Ce module permet de définir une interface textuelle simple qui servira à tester cette première version du jeu.

Complétez l'ensemble des fonctions de ce module en vous appuyant sur des fonctions type printf pour effectuer les affichages demandés (voir documentation Doxygen disponible dans *board_view.h*).

Remarque pour la construction

Notez qu'il faut ajouter la définition de CONFIG_TEXTUI dans les options de construction du programme pour activer correctement la compilation conditionnelle.

4.5 Module *player_manager_mock*

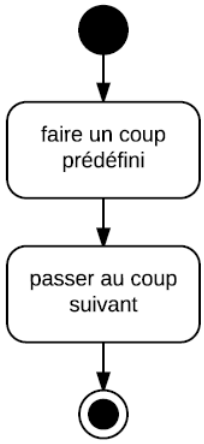

FIGURE 4 – Algorithme de oneTurn

Objectif Proposer un jeu de tests pour valider la logique du jeu.

- En s'inspirant des tests de l'étape 0, proposer un jeu de tests qui représente une ou plusieurs parties du jeu.
- Écrivez la fonction PlayerManager_oneTurn pour jouer un tour du jeu de tests que vous avez défini (diagramme de la figure 4). Cette fonction s'appuie sur le module *board* pour jouer un coup prédéfini (figure 3).

Remarque pour la construction

Activez la définition de CONFIG_PLAYER_MANAGER_MOCK dans les options de construction du programme pour activer correctement la compilation conditionnelle.

4.6 Module *Game*

Écrivez le corps des fonctions Game_init, Game_free et Game_loop. Cette dernière s'appuie sur la fonction PlayerManager_oneTurn pour jouer chaque coup.

4.7 Récapitulatif des actions à réaliser

Vérifiez que vous avez bien réalisé chacune des actions de cette étape.

Action	Fait
Board_putPiece	
Appel des SquareChangeCallback et EndOfGameCallback dans la fonction Board_putPiece	
Board_squareContent	
Board_init	
Board_free	
Module board_view_text	
Module player_manager_mock	
Module Game	
Validation de cette mouture du jeu de Morpion	

5 Etape 2

Le moteur de jeu étant à présent correctement réalisé, vous allez développer l'aspect interactif du jeu. Dans un premier temps, l'interface reste en mode texte. Lutilisateur devra donc saisir les coordonnées de la case où il souhaite jouer.

Vous trouverez la base du code source de létape 2 sur le campus, à ajouter à votre projet. Il faudra bien évidemment valider votre travail avant de passer à létape suivante.

Remarque pour la construction

Après avoir intégré les fichiers, il faut adapter les paramètres de construction de manière à définir CONFIG_PLAYER_MANAGER_SCANF à la place de CONFIG_PLAYER_MANAGER_MOCK.

5.1 Fonction PlayerManager_oneTurn

Cette fonction, lorsqu'elle est appelée, doit réaliser un tour de jeu. Cela implique donc qu'elle soit à l'écoute des actions de l'utilisateur.

Dans le cas présent, l'utilisateur devra saisir les coordonnées de la case en respectant les définitions suivantes :

- le format de saisie est rigoureusement " X,Y "; il n'y a notamment pas d'espace ni d'autre séparateur que la virgule entre les deux coordonnées ;
- X est le numéro de colonne dans le tableau ;
- Y est le numéro de la ligne dans le tableau ;
- les numéros (de ligne ou de colonne) sont compris entre 0 et 2 inclus.

À partir des coordonnées saisies, il faut utiliser le module *Board* afin de placer effectivement un pion dans la case correspondante. En cas de saisie incorrecte (non respect des définitions ci-dessus ou placement sur une case non vide), la fonction devra attendre une nouvelle saisie de l'utilisateur.

5.2 Validation

Vérifiez que votre programme, avec la saisie utilisateur :

— respecte toujours le cahier des charges : pas de régression sur le fonctionnement de la logique de jeu ;
— fonctionne pour toutes les saisies utilisateurs nominales (auxquelles on peut s'attendre) ;
— ne dysfonctionne pas en cas de saisie non nominale (robustesse).

5.3 Récapitulatif des actions à réaliser

Vérifiez que vous avez bien réalisé chacune des actions de cette étape.

Action	Fait
Module player_manager_scanf Validation de la saisie utilisateur	

6 Etape 3

Nous souhaitons à présent donner un peu d'allure à notre jeu en utilisant une bibliothèque permettant de construire une interface graphique : la bibliothèque SDL2 [2].

Une capture d'écran du résultat escompté est présentée en figure 1. Elle est basée sur des images (fournies sur le campus, avec les fichiers de l'étape 3) que vous pourrez adapter à votre goût.

L'essentiel du code spécifique à l'utilisation de la bibliothèque SDL est déjà fourni. Il reste cependant des parties très spécifiques à l'application courante, telles que les dimensions graphiques.

Le code source de l'étape 3 est à ajouter à votre projet. Nous allons, dans un premier temps, réaliser le module board_view_sdl avant de prendre en compte les interactions utilisateur dans le module square_manager_sdl.

Remarque pour la construction

En vous inspirant de ce qui a été fait précédemment et du code source, activez la compilation conditionnelle pour le module board_view_sdl.

6.1 Fonction BoardView_displayAll

En utilisant la fonction privée renderImage, affichez l'image de fond préchargée dans la variable globale BackgroundImage.

Ensuite, en appelant la fonction BoardView_displaySquare, affichez chacune des cases du tableau. Le résultat ne sera cependant visible qu'une fois cette dernière codée (paragraphe 6.2).

Testez et validez le bon fonctionnement de l'affichage du fond en exécutant le programme.

6.2 Fonction BoardView_displaySquare

En utilisant la spécification des dimensions données en figure 5 et la fonction renderImage, implémentez la fonction BoardView_displaySquare pour placer la bonne image (chargées dans les variables SpriteO et SpriteX) au bon endroit dans la fenêtre, par rapport aux coordonnées logiques fournies.

Validez les points suivants :

2. http ://www.libsdl.org/

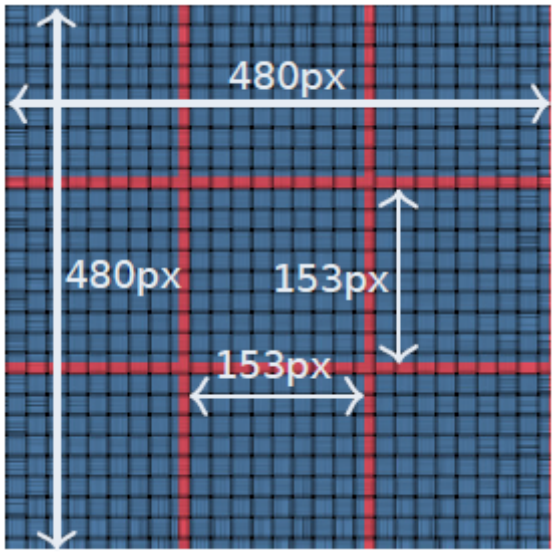

FIGURE 5 – Dimensions des images du jeu

— l'image s'affiche dans la bonne case ;
— l'image correspond à la spécification ;
— l'image est positionnée correctement (au pixel près).

6.3 Fermeture de la fenêtre

Remarque pour la construction

En vous inspirant de ce qui a été fait précédemment et du code source, activez la compilation conditionnelle pour le module player_manager_sdl.

Dans un premier temps, pour des raisons pratiques, nous allons faire en sorte de terminer l'application lors d'une demande de fermeture de la fenêtre.

Pour rappel, la fonction PlayerManager_oneTurn attend la saisie de l'utilisateur (SDL_waitEvent).

Le type d'événement SDL_QUIT est une demande de fermeture de fenêtre. Adaptez le cas de figure correspondant pour quitter l'application.

6.4 Prise en compte du clic souris

En vous aidant de la documentation de la bibliothèque SDL2 et plus particulièrement du type SDL_EvenType, créez un nouveau cas de figure dans la fonction PlayerManager_oneTurn pour prendre en compte un clic de souris.

Appuyez-vous enfin sur l'algorithme présenté en figure 6 pour terminer l'écriture de la fonction PlayerManager_oneTurn.

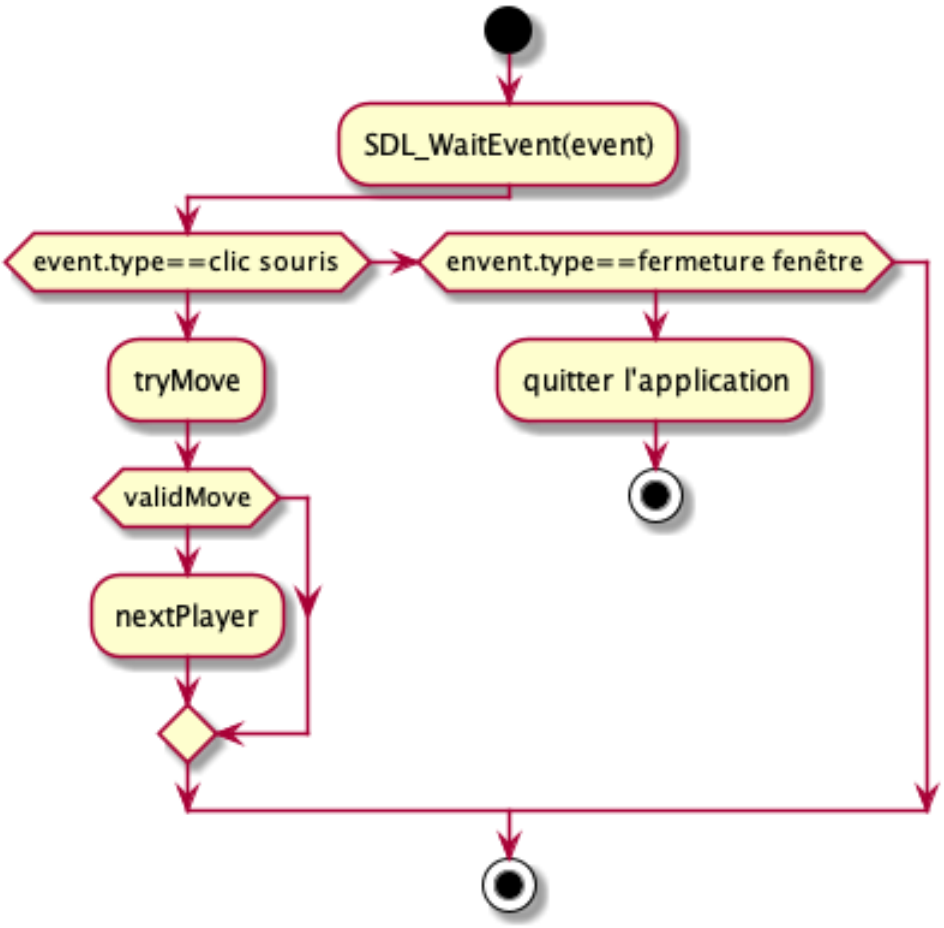

FIGURE 6 – Algorithme de la fonction PlayerManager_oneTurn

6.5 Fonction tryMove

Pour coder cette fonction, il est nécessaire de convertir correctement des coordonnées du clic dans la fenêtre en coordonnées logiques d'une case dans le tableau.

Une fois la conversion effectuée, si le clic est dans une case, il faut appeler la fonction Board_putPiece pour placer effectivement un pion. En cas de placement de pion invalide, il faut appeler la fonction BoardView_sayCannotPutPiece pour avertir l'utilisateur.

6.6 Validation

Validez le fonctionnement de votre application, particulièrement son :

— comportement lors du clic à lintérieur, proche du centre dune case ;
— comportement lors du clic à lintérieur, proche de lextérieur dune case ;
— comportement lors du clic dans lespace entre des cases.

6.7 Récapitulatif des actions à réaliser

Vérifiez que vous avez bien réalisé chacune des actions de cette étape.

Action	Fait
paramétrage de la compilation conditionnelle pour le module board_view_sdl	
codage de la fonction BoardView_displayAll	
codage de la fonction BoardView_displaySquare	
paramétrage de la compilation conditionnelle pour le module player_manager_sdl	
gestion de la fermeture de la fenêtre	
interprétation du clic souris	
action conséquente au clic souris	
validation générale du jeu	

7 Aller plus loin. . .

Vous pouvez améliorer les aspects graphiques du jeu, notamment implémenter les fonctions comme BoardView_displayPlayersTurn en SDL2.

Mais vous devriez désormais être prêts à réaliser votre propre jeu (ou autre application similaire) en vous inspirant de ce ce projet. Vous pouvez démarrer la réalisation de votre "projet libre" en commençant par faire valider votre sujet par un enseignant.