# Rapport du projet de C++

## Projet

Ce projet implémente les trois jeux de plateau suivants:

- Butin
- Gounki
- Safari

L'ensemble du programme utilise l'interface graphique SFML, la console étant utile seulement pour suivre l'état de création et destruction des objets.

## Aspects significatifs

idées: ~~MVC pattern~~, ~~tentative d'utilisation des templates~~, ~~abstraction facilitant l'implémentation d'un jeu~~, d'une interface, du controller, ~~gameconfig~~, *héritage avec private/protected*, ~~blocage des copies~~, ~~les boucles ne font pas de copies~~

Le projet est basé sur le patron `Modèle-Vue-Contrôleur`, nous permettant de séparer efficacement les différents aspects du projet.
Lors de la conception nous avons décidé d'effectuer le plus d'abstraction possible dans chacunes des branches de ce patron afin de faciliter l'implémentation d'un nouveau jeu de plateau, peu importe sa taille et son nombre de joueurs.

Le modèle ainsi que la vue contiennent dans leur dossier racine respectif plusieurs classes étant composées d'interfaces et de classes abstraites contenant les fonctions à redéfinir afin d'adapter un nouveau jeu.
Cette abstraction du modèle nous permet d'établir un niveau d'abstraction significatif de la vue, permettant d'appeler les mêmes fonctions de la classe `Game` sans dépendre d'une des fonctions propres à un jeu.
Certaines fonctions n'étant pas `pure virtual` sont définies dans le fichier `cpp` correspondant et permettent de limiter les répétitions de code dans les fichiers implémentés. Il est toujours possible de redéfinir la fonction si le mode de jeu nécessite une interprétation plus spécifique.
L'interface `GameConfig` est une classe particulière permettant de généraliser les configurations et phases d'initialisation des différents jeux. Une fonction de `Game` prenant une instance de cette interface en argument permet d'initialiser chaque jeu avec des attributs différents.

Une implémentation de notre projet par l'utilisation des `templates` a été essayée, permettant de rendre le code plus compréhensible et limitant les `cast` dans le contenu des fonctions.

Malheureusement cette implémentation faisait remonter l'abstraction jusqu'à la vue, rendant impossible d'avoir un menu de sélection de jeu via SFML.
Notre implémentation actuelle utilise les classes abstraites et interfaces comme attributs et dépend du polymorphisme afin de définir les classes effectives.

Une attention particulière a été mise sur les copies. Nous avons décidé de bloquer les copies de l'ensemble des classes du modèle et de la vue mise à part l'interface `GameConfig`.
Le code effectue un nombre minimal de copies et favorise l'utilisation de référence si la logique lui permet.

Le projet dispose d'un héritage strict sécurisant l'accès aux objets et fonctions qui composent `Game`. Seulement les fonctions et membres publics sont visibles, tout le reste des attributs et des fonctions ne sont pas accessibles depuis l'extérieur.

En ce qui concerne l'affichage, nous avons décidé de baser la construction des différentes fenêtres graphiques sur le concept de pile, où l'élément au sommet de la pile est celui qui sera dessiné sur l'écran de l'utilisateur. En plus d'une classe pour gérer quelle fenêtre se trouve dans notre pile de States nous avons 2 classes dont le contrôleur possède un pointeur pour stocker des Textures et des méthodes utilisées pour interagir avec les Sprite de SFML.

L'interface "States" représente ces fenêtre à l'équivalent de d'un JPanel en swing et pour le States dédié à l'affichage et le déroulement des jeux c'est GameState qui permet d'offrir un certain niveau d'abstraction malgré le fait qu certaine méthodes est le besoin d'être redéfinies d'un jeux à l'autre.

Le projet repose aussi sur le `header` `SETTINGS.hpp` contenant des macros permettant de paramétrer l'affichage graphique ainsi que les jeux, offrant plus de modularité.

## Problèmes connus

La configuration de la résolution d'écran est fonctionnelle et l'interface s'adapte bien, cependant certaines images de l'interface n'ont pas pû être adaptées à temps.

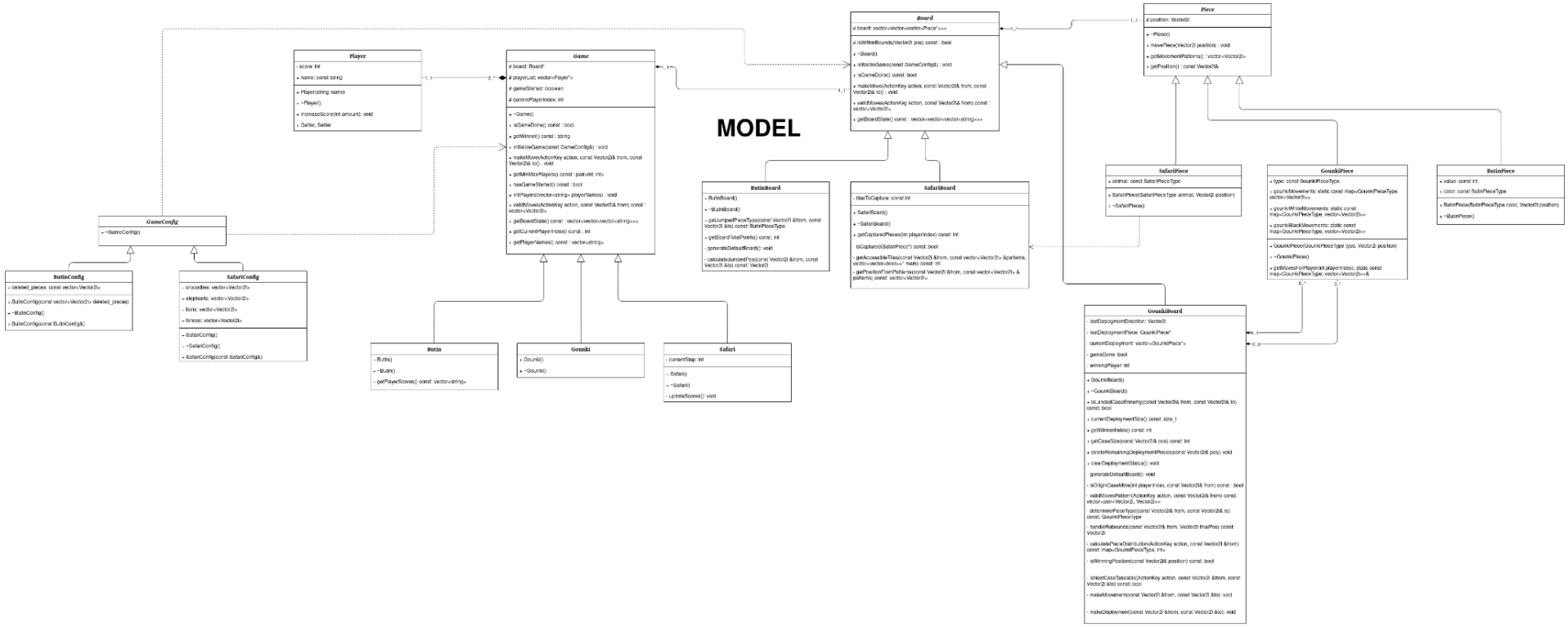
Lors du lancement du programme, il est possible que la souris se retrouve coincée dans une zone invisible et ne puisse pas se déplacer sur l'intégralité de l'écran. Mettre la fenêtre en arrière-plan (ALT+TAB ou autre) et revenir résout le problème.

## Extensions

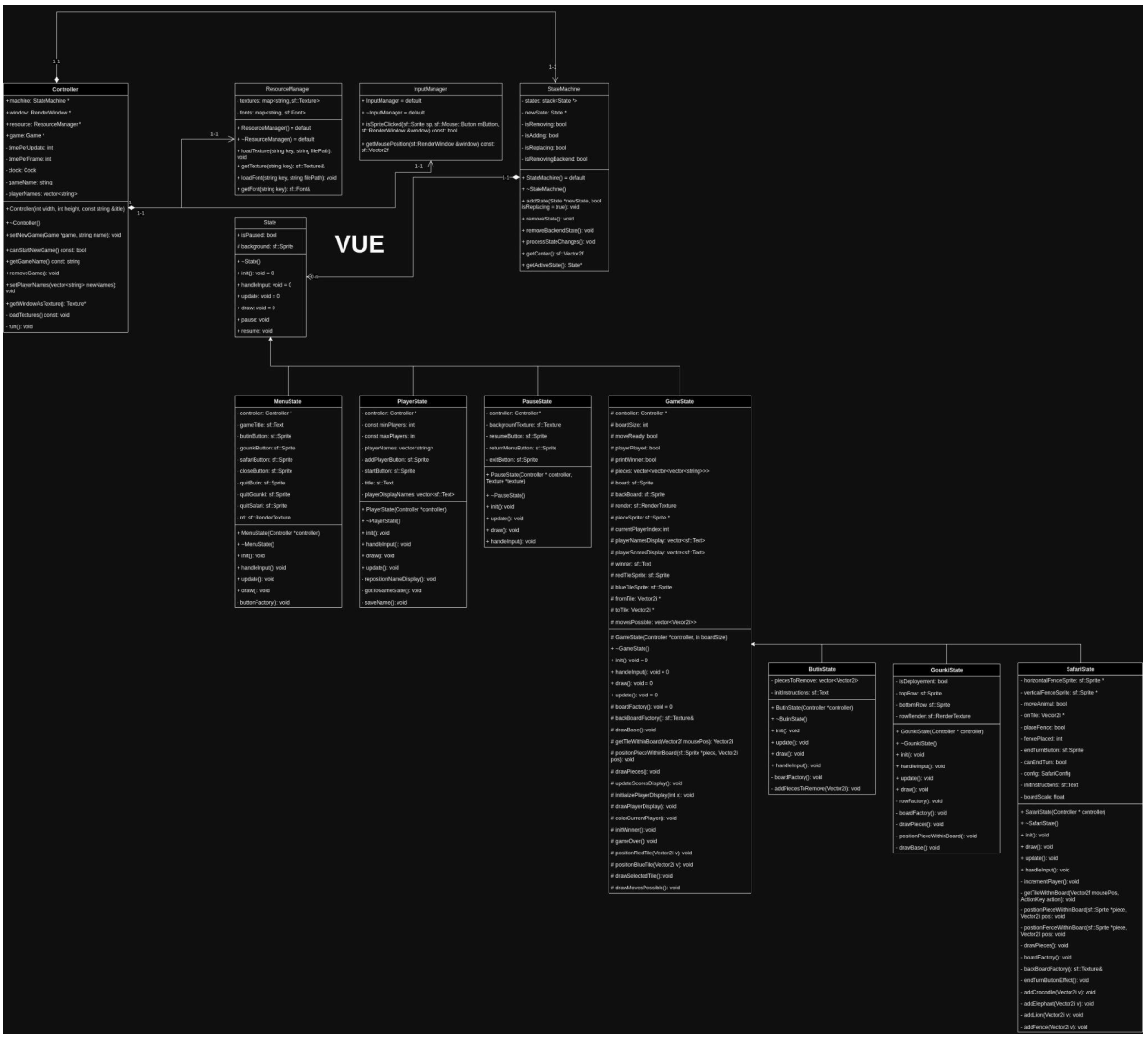
Aucune extension n'a été implémentée.

## UML

## Model:

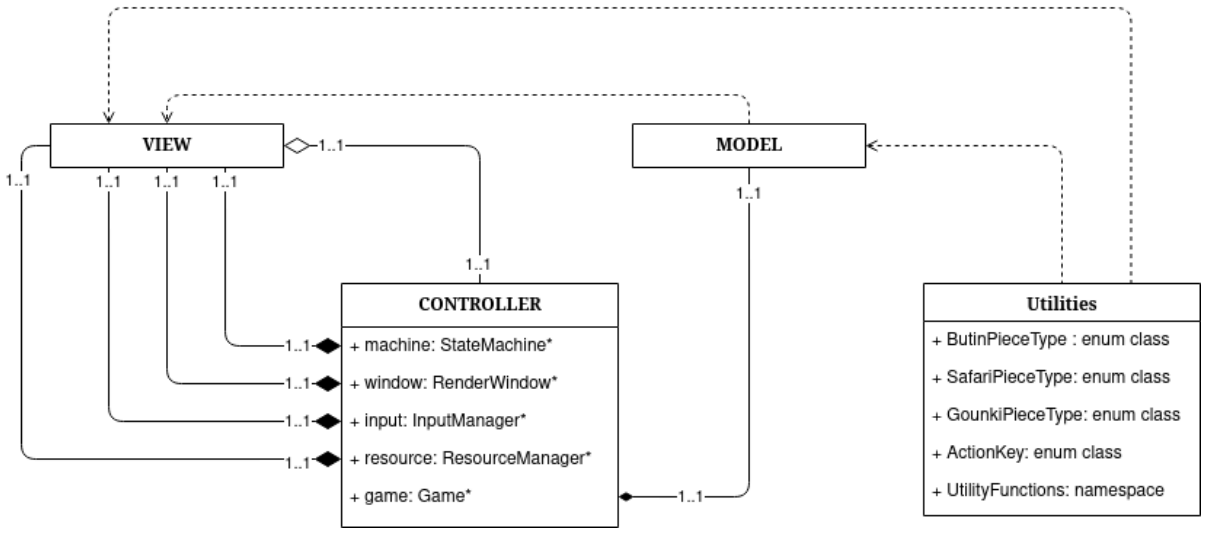

MODEL
Player
- score: Int
+ name: const string
+ Player(string name)
+ ~Player()
+ increaseScore(int amount): void
+ Getter, Setter
Game
# board: Board*
# playerList: vector<Player*>
# gameStarted: boolean
# currentPlayerIndex: int
+ ~Game()
+ isGameDone(): bool
+ getWinner(): string
+ initializeGame(const GameConfig&) : void
+ makeMove(ActionKey action, const Vector2i& from, const Vector2i& to) : void
+ getMinMaxPlayers(): pair<int, int>
+ hasGameStarted(): bool
+ initPlayers(vector<string> playerNames) : void
+ validMoves(ActionKey action, const Vector2i& from) const : vector<Vector2i>
+ getBoardState(): const : vector<vector<vector<string>>>
+ getCurrentPlayerIndex(): const : int
+ getPlayerNames(): const : vector<string>
Board
# board: vector<vector<vector<Piece*>>>
# isWithinBounds(Vector2i pos) const : bool
+ ~Board()
+ initializeGame(const GameConfig&) : void
+ isGameDone(): const: bool
+ makeMove(ActionKey action, const Vector2i& from, const Vector2i& to) : void
+ validMoves(ActionKey action, const Vector2i& from) const : vector<Vector2i>
+ getBoardState(): const : vector<vector<vector<string>>>
Piece
# position: Vector2i
+ ~Piece()
+ movePiece(Vector2i position) : void
+ getMovementPatterns(): vector<Vector2i>
+ getPosition(): const Vector2i&
GameConfig
+ ~GameConfig()
ButinConfig
- deleted_pieces: const vector<Vector2i>
+ ButinConfig(const vector<Vector2i> deleted_pieces)
+ ~ButinConfig()
+ ButinConfig(const ButinConfig&)
SafariConfig
- crocodiles: vector<Vector2i>
+ elephants: vector<Vector2i>
- lions: vector<Vector2i>
+ rinoces: vector<Vector2i>
+ SafariConfig()
- ~SafariConfig()
+ SafariConfig(const SafariConfig&)
Butin
- Butin()
+ ~Butin()
+ getPlayerScores(): const: vector<string>
Gounki
+ Gounki()
+ ~Gounki()
Safari
- currentStep: int
- Safari()
+ ~Safari()
- updateScores(): void
ButinBoard
+ ButinBoard()
+ ~ButinBoard()
+ getJumpedPieceType(const Vector2i &from, const Vector2i &to) const: ButinPieceType
+ getBoardTotalPoints(): const: int
- generateDefaultBoard(): void
- calculateJumpedPos(const Vector2i &from, const Vector2i &to) const: Vector2i
SafariBoard
- tilesToCapture: const int
+ SafariBoard()
+ ~SafariBoard()
+ getCapturedPieces(int playerIndex): const: int
- isCaptured(SafariPiece*): const: bool
- getAccessibleTiles(const Vector2i &from, const vector<Vector2i> &patterns, vector<vector<bool>>* mark): const: int
- getPositionFromPattern(const Vector2i &from, const vector<Vector2i> & patterns): const: vector<Vector2i>
SafariPiece
+ animal: const SafariPieceType
+ SafariPiece(SafariPieceType animal, Vector2i position)
+ ~SafariPiece()
GounkiBoard
- lastDeploymentDirection: Vector2i
- lastDeploymentPiece: GounkiPiece*
- currentDeployment: vector<GounkiPiece*>
- gameDone: bool
- winningPlayer: int
+ GounkiBoard()
+ ~GounkiBoard()
+ isLandedCaseEnnemy(const Vector2i& from, const Vector2i& to) const: bool
+ currentDeploymentSize(): const: size_t
+ getWinnerIndex(): const: int
+ getCaseSize(const Vector2i& pos) const: int
+ deleteRemainingDeploymentPieces(const Vector2i& pos): void
+ clearDeploymentStatus(): void
- generateDefaultBoard(): void
- isOriginCaseMine(int playerIndex, const Vector2i& from) const : bool
- validMovesPattern(ActionKey action, const Vector2i& from) const: vector<pair<Vector2i, Vector2i>>
- determinePieceType(const Vector2i& from, const Vector2i& to) const: GounkiPieceType
- handleRebounds(const Vector2i& from, Vector2i finalPos) const: Vector2i
- calculatePieceDistributions(ActionKey action, const Vector2i &from) const: map<GounkiPieceType, int>
- isWinningPosition(const Vector2i& position) const: bool
- isNextCaseTakeable(ActionKey action, const Vector2i &from, const Vector2i &to) const: bool
- makeMovement(const Vector2i &from, const Vector2i &to): void
- makeDeployment(const Vector2i &from, const Vector2i &to): void
GounkiPiece
+ type: const GounkiPieceType
- gounkiMovements: static const map<GounkiPieceType, vector<Vector2i>>
+ gounkiWhiteMovements: static const map<GounkiPieceType, vector<Vector2i>>
+ gounkiBlackMovements: static const map<GounkiPieceType, vector<Vector2i>>
+ GounkiPiece(GounkiPieceType type, Vector2 position)
+ ~GounkiPiece()
+ getMovesForPlayer(int playerIndex): static const map<GounkiPieceType, vector<Vector2i>>&
ButinPiece
+ value: const int
+ color: const ButinPieceType
+ ButinPiece(ButinPieceType color, Vector2i position)
+ ~ButinPiece()

# View:




# Controller:

# Model-View-Controller:




**PICKERN YANN 22012875**
**RODRIGUEZ Lucas 22002335**