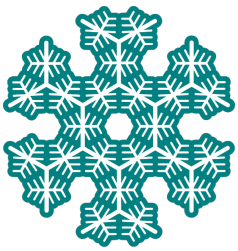
Corentin VÉROT

Projet de programmation

420-P66-SI



Projet Yams

Charte de projet

Travail présenté à
Martin Carignan

Département de l'informatique
CÉGEP de Sept-Îles
15 mai 2020

Table des matières

1 Introduction **1**

2 Présentation du livrable **2**

2.1 Consignes d'utilisation 3

3 Réalisation du projet **5**

4 Conception logicielle **7**

5 Vision robotique **11**

5.0.1 Algorithme 1 : Détection par aire minimum 11

5.0.2 Algorithme 2 : Détection par blob / goutte 12

6 Annexe **13**

6.1 Version des logiciels et bibliothèques utilisés (en mai 2020) 13

Bibliographie **14**

1. Introduction

Ce document porte sur le projet de Yams qui fait suite à l'annulation du projet sur le drone DJI Matrice 100 pour cause de COVID-19.

Ce projet vise à créer une application qui permette de comptabiliser les points durant une partie de Yams en reconnaissant les dès, tout en conseillant le mieux possible les joueurs en proposant les coups à réaliser les plus efficaces.

Feuille de Score www.regles-de-jeux.com

JOUEURS						
Total de 1						
Total de 2						
Total de 3						
Total de 4						
Total de 5						
Total de 6						
Total						
Si total I > 63 alors Bonus de 35 points						
Total partie intermédiaire						
Brelan (Total des 3 dés)						
Carré (Total des 4 dés)						
Full (25 points)						
Petite Suite (30 points)						
Grande Suite (40 points)						
Yams (50 points)						
Chance (Total des 5 dés)						
Total II						
TOTAL						

Exemple de feuille de score

Le jeu doit :

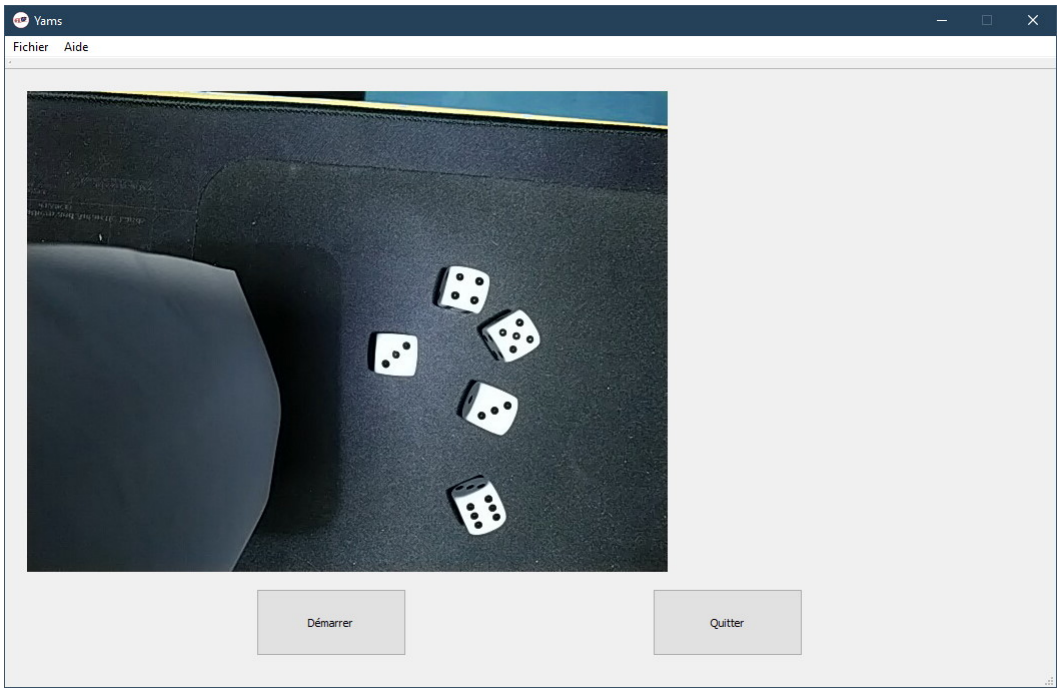
- proposer au joueur les 3 meilleures possibilités de coup en fonction des dès qu'il a lancé ;
- remplir la feuille de score en fonction du choix de l'utilisateur ;
- proposer une expérience de jeux conviviale et agréable et sans bogue ;
- être exécutable sur n'importe quel ordinateur disposant d'une caméra.

La caméra doit déterminer :

- la position X et Y des dés ;
- la valeur de chacun des dés.

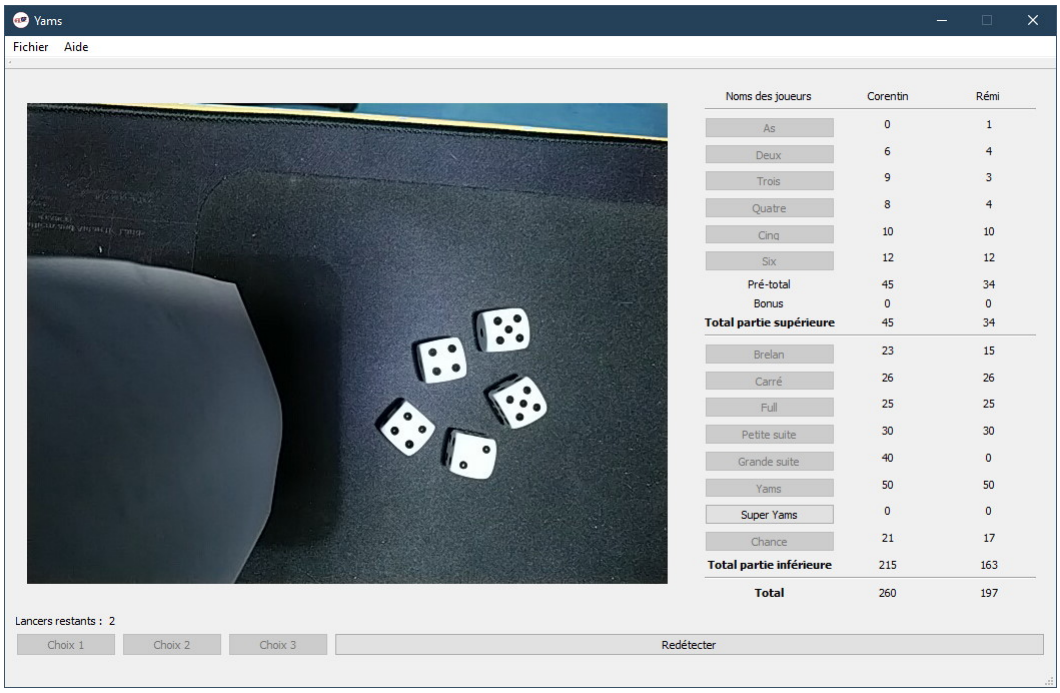
La date limite de remise du projet est fixée au 15 Mai 2020.

2. Présentation du livrable



Accueil

Le jeu prend la forme d'une fenêtre présentant les scores des différents joueurs sous la forme d'un tableau sur la droite, le rendu caméra sur la gauche et les meilleurs choix en bas de la fenêtre.
Si l'utilisateur ne trouve pas les choix proposés judicieux, il peut néanmoins choisir l'action qu'il souhaite faire en cliquant sur le bouton correspondant à l'action qu'il désire réaliser dans la colonne gauche du tableau des scores. Les boutons des actions déjà réalisées seront grisés.



En cours de partie

Un rappel des règles du jeu se trouve dans le menu *Aide - Règles du jeu*.

Le Yams étant un jeux se jouant avec un nombre minimal de deux joueurs sans maximum, il est possible de jouer avec l'application avec autant de contraintes.

La reconnaissance d'image servant à détecter les dés peut ne pas être infaillible. C'est pourquoi un bouton *Redétecter* a été disposé à droite des boutons représentant les meilleurs choix.

2.1 Consignes d'utilisation

Pour un fonctionnement optimal de la reconnaissance d'image, merci de :

- Jouer sur une surface sombre ;
- Placer une caméra au dessus du jeu à 20 cm de hauteur ;
- Éviter les reflet et ombres ;
- Jouer dans un environnement lumineux ;
- Faire en sorte de mettre hors champ toute zone de mouvement en dehors du plateau de jeu ;
- Bien positionner les dés en dessous de l'objectif de la caméra ;

- Maintenir les dés à environ 1cm de distance les uns des autres. [1]

[1] Les règles de distanciation sociale en période de pandémie s'appliquant aussi aux dés.

3. Réalisation du projet

Ce projet a été réalisé en C++ avec le Framework Qt dans sa version 5.12.6 (pour l'interface utilisateur) et la bibliothèque OpenCV en version 4.3.0 (pour la reconnaissance d'image). Pour un portage facile d'une plate-forme à l'autre, le projet a été conçu avec CMake.

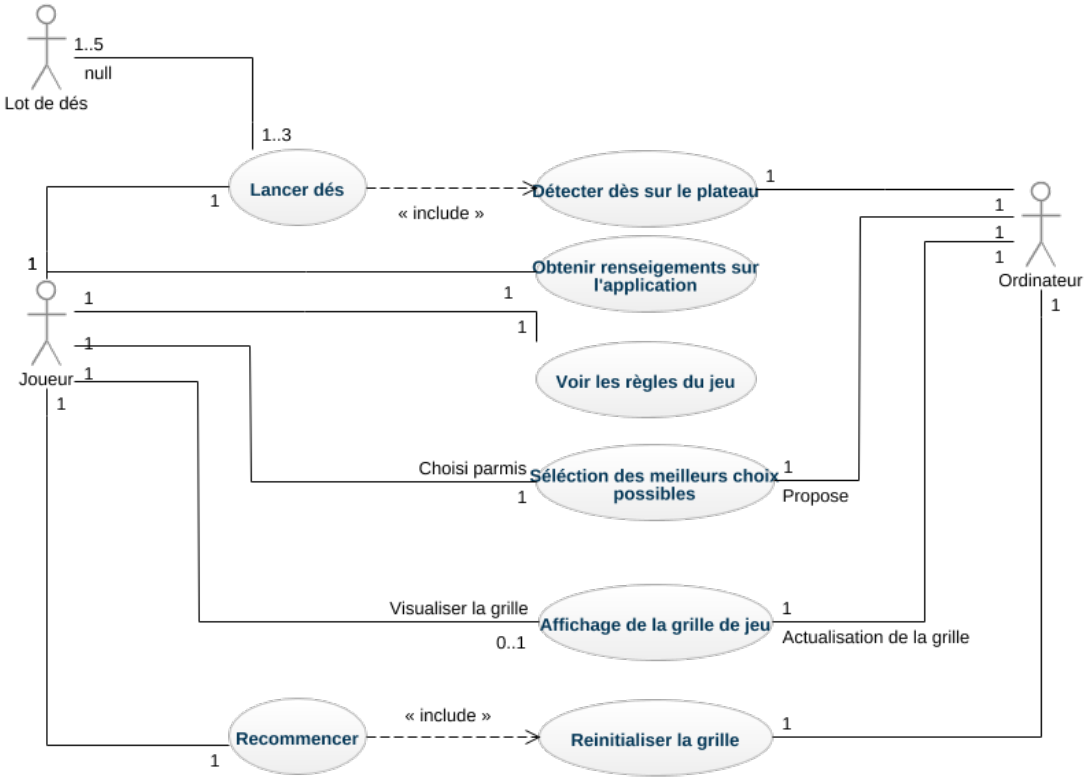
Suivi de projet Yams :

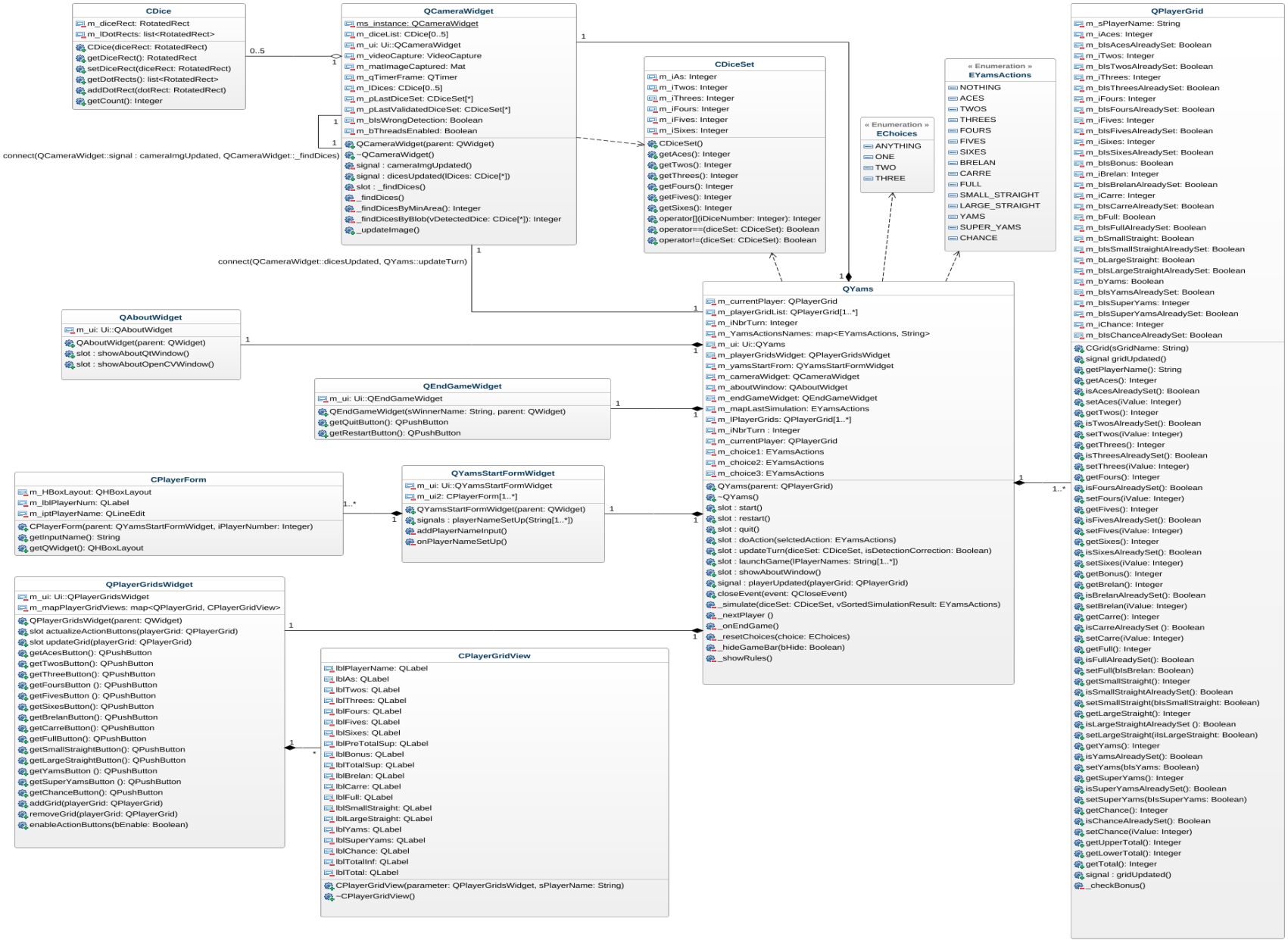
Objet	Temps théorique	Dates théoriques	Temps réel	Dates réelles
I - Mise en place de l'environnement de développement	**4**	04/11 → 04/14	**1,5**	04/11 → 04/14
Installation d'OpenCV	3		1	
Mise en place de Doxygen pour documentation en ligne et LaTeX	1		0,5	
II – Formation et prise de connaissance avec OpenCV	**8**	04/14 → 04/20	**10**	04/14 → 04/17
Réalisation d'opérations sur les images	8		10	
III – Conception du logiciel	**10**	04/20 → 04/25	**6**	04/18 → 04/21
Diagramme des cas d'utilisation	1		0,5	
Diagramme d'objets	1		0,5	
Diagramme d'interractions	2		1	
Diagramme de classes	4		3	
Esquisse de l'interface utilisateur	2		1	
IV – Détection des dès	**12**	04/25 → 05/01	**8**	04/21 → 04/24
Détection d'un dès quelconque	6		5	
Détection des tâches sur les dès et comptage	6		3	
V – Création du jeu	**26**	05/01 → 05/15	**22**	04/24 → 05/10
Codage des modèles	6		4	
Codage des vues	4		4	
Codage de la jouabilité	8		7	
Codage de l'IA	4		3	
Correction des bogues	4		4	
Total :	**60**	Total réel :	**47,5**	

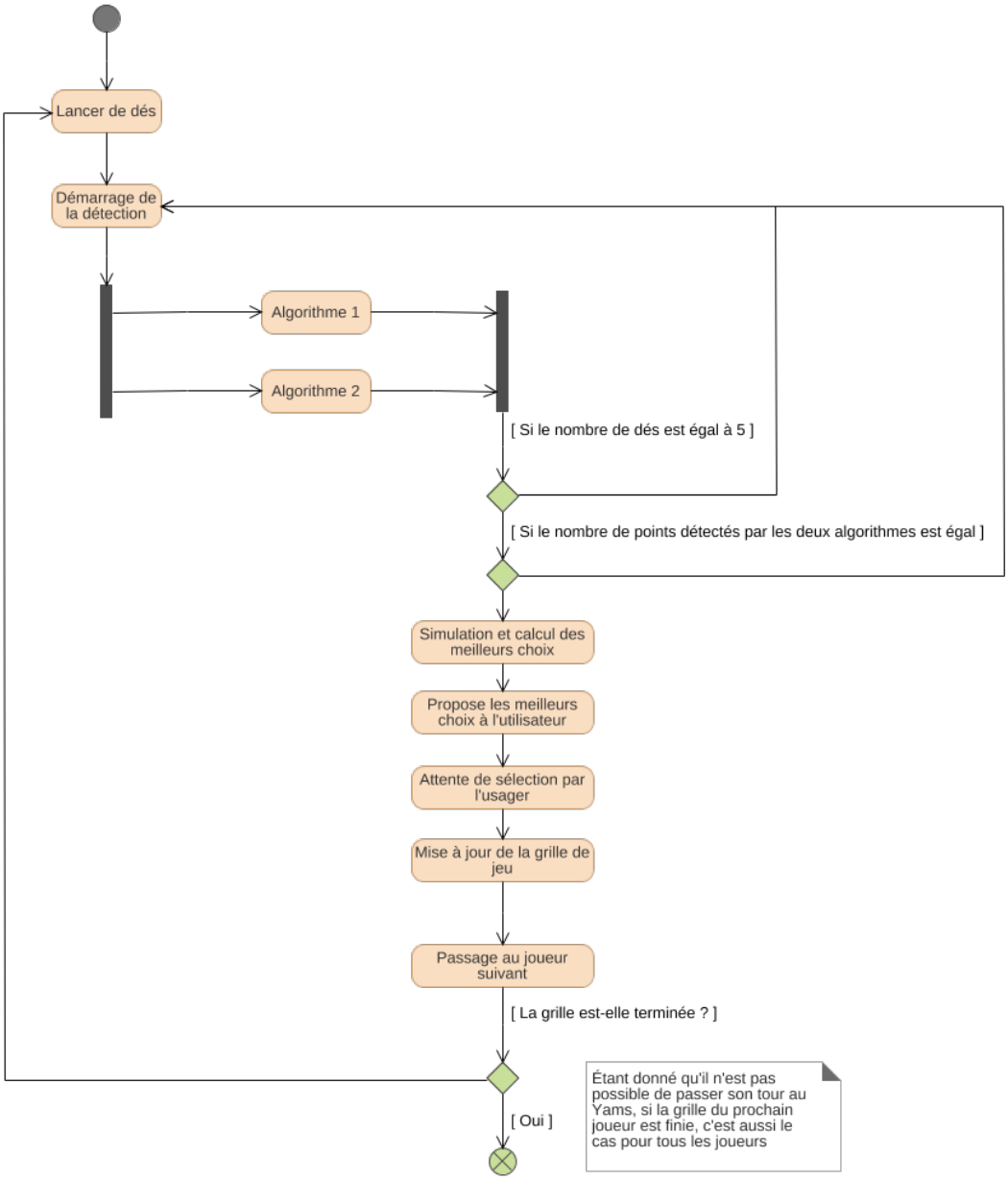
4. Conception logicielle

Les vues ne sont pas représentées sur le diagramme de classes étant donnée qu'elles ont été conçu avec Qt Designer.

Pour une meilleure clarté, vous pouvez retrouver l'intégralité de la conception en images png dans le dépôt GitHub du projet[5].


Lot de dés
1..5
null
1..3
Lancer dés
1
« include »
Détecter dès sur le plateau
1
1
1
1
Ordinateur
1
Obtenir renseigements sur l'application
1
1
1
Joueur
1
1
1
1
Voir les règles du jeu
Choisi parmis
1
Séléction des meilleurs choix possibles
1
Propose
Visualiser la grille
0..1
Affichage de la grille de jeu
1
Actualisation de la grille
« include »
Recommencer
1
Reinitialiser la grille
1


CDice
m_diceRect: RotatedRect
m_lDotRects: list<RotatedRect>
CDice(diceRect: RotatedRect)
getDiceRect(): RotatedRect
setDiceRect(diceRect: RotatedRect)
getDotRects(): list<RotatedRect>
addDotRect(dotRect: RotatedRect)
getCount(): Integer
0..5
1
QCameraWidget
ms_instance: QCameraWidget
m_diceList: CDice[0..5]
m_ui: Ui::QCameraWidget
m_videoCapture: VideoCapture
m_matImageCaptured: Mat
m_qTimerFrame: QTimer
m_lDices: CDice[0..5]
m_pLastDiceSet: CDiceSet[*]
m_pLastValidatedDiceSet: CDiceSet[*]
m_blsWrongDetection: Boolean
m_bThreadsEnabled: Boolean
QCameraWidget(parent: QWidget)
~QCameraWidget()
signal : cameraImgUpdated()
signal : dicesUpdated(lDices: CDice[*])
slot : _findDices()
_findDices()
_findDicesByMinArea(): Integer
_findDicesByBlob(vDetectedDice: CDice[*]): Integer
_updateImage()
connect(QCameraWidget::signal : cameraImgUpdated, QCameraWidget::_findDices)
connect(QCameraWidget::dicesUpdated, QYams::updateTurn)
CDiceSet
m_iAs: Integer
m_iTwos: Integer
m_iThrees: Integer
m_iFours: Integer
m_iFives: Integer
m_iSixes: Integer
CDiceSet()
getAces(): Integer
getTwos(): Integer
getThrees(): Integer
getFours(): Integer
getFives(): Integer
getSixes(): Integer
operator[](iDiceNumber: Integer): Integer
operator==(diceSet: CDiceSet): Boolean
operator!=(diceSet: CDiceSet): Boolean
« Enumeration »
EChoices
ANYTHING
ONE
TWO
THREE
« Enumeration »
EYamsActions
NOTHING
ACES
TWOS
THREES
FOURS
FIVES
SIXES
BRELAN
CARRE
FULL
SMALL_STRAIGHT
LARGE_STRAIGHT
YAMS
SUPER_YAMS
CHANCE
QPlayerGrid
m_sPlayerName: String
m_iAces: Integer
m_blsAcesAlreadySet: Boolean
m_iTwos: Integer
m_blsTwosAlreadySet: Boolean
m_iThrees: Integer
m_blsThreesAlreadySet: Boolean
m_iFours: Integer
m_blsFoursAlreadySet: Boolean
m_iFives: Integer
m_blsFivesAlreadySet: Boolean
m_iSixes: Integer
m_blsSixesAlreadySet: Boolean
m_blsBonus: Boolean
m_iBrelan: Integer
m_blsBrelanAlreadySet: Boolean
m_iCarre: Integer
m_blsCarreAlreadySet: Boolean
m_bFull: Boolean
m_blsFullAlreadySet: Boolean
m_bSmallStraight: Boolean
m_blsSmallStraightAlreadySet: Boolean
m_bLargeStraight: Boolean
m_blsLargeStraightAlreadySet: Boolean
m_bYams: Boolean
m_blsYamsAlreadySet: Boolean
m_blsSuperYams: Integer
m_blsSuperYamsAlreadySet: Boolean
m_iChance: Integer
m_blsChanceAlreadySet: Boolean
CGrid(sGridName: String)
signal gridUpdated()
getPlayerName(): String
getAces(): Integer
isAcesAlreadySet(): Boolean
setAces(iValue: Integer)
getTwos(): Integer
isTwosAlreadySet(): Boolean
setTwos(iValue: Integer)
getThrees(): Integer
isThreesAlreadySet(): Boolean
setThrees(iValue: Integer)
getFours(): Integer
isFoursAlreadySet(): Boolean
setFours(iValue: Integer)
getFives(): Integer
isFivesAlreadySet(): Boolean
setFives(iValue: Integer)
getSixes(): Integer
isSixesAlreadySet(): Boolean
setSixes(iValue: Integer)
getBonus(): Integer
getBrelan(): Integer
isBrelanAlreadySet(): Boolean
setBrelan(iValue: Integer)
getCarre(): Integer
isCarreAlreadySet (): Boolean
setCarre(iValue: Integer)
getFull(): Integer
isFullAlreadySet(): Boolean
setFull(blsBrelan: Boolean)
getSmallStraight(): Integer
isSmallStraightAlreadySet(): Boolean
setSmallStraight(blsSmallStraight: Boolean)
getLargeStraight(): Integer
isLargeStraightAlreadySet (): Boolean
setLargeStraight(ilsLargeStraight: Boolean)
getYams(): Integer
isYamsAlreadySet(): Boolean
setYams(blsYams: Boolean)
getSuperYams(): Integer
isSuperYamsAlreadySet(): Boolean
setSuperYams(blsSuperYams: Boolean)
getChance(): Integer
isChanceAlreadySet(): Boolean
setChance(iValue: Integer)
getUpperTotal(): Integer
getLowerTotal(): Integer
getTotal(): Integer
signal : gridUpdated()
_checkBonus()
QYams
m_currentPlayer: QPlayerGrid
m_playerGridList: QPlayerGrid[1..*]
m_iNbrTurn: Integer
m_YamsActionsNames: map<EYamsActions, String>
m_ui: Ui::QYams
m_playerGridsWidget: QPlayerGridsWidget
m_yamsStartFrom: QYamsStartFormWidget
m_cameraWidget: QCameraWidget
m_aboutWindow: QAboutWidget
m_endGameWidget: QEndGameWidget
m_mapLastSimulation: EYamsActions
m_lPlayerGrids: QPlayerGrid[1..*]
m_iNbrTurn : Integer
m_currentPlayer: QPlayerGrid
m_choice1: EYamsActions
m_choice2: EYamsActions
m_choice3: EYamsActions
QYams(parent: QPlayerGrid)
~QYams()
slot : start()
slot : restart()
slot : quit()
slot : doAction(selctedAction: EYamsActions)
slot : updateTurn(diceSet: CDiceSet, isDetectionCorrection: Boolean)
slot : launchGame(lPlayerNames: String[1..*])
slot : showAboutWindow()
signal : playerUpdated(playerGrid: QPlayerGrid)
closeEvent(event: QCloseEvent)
_simulate(diceSet: CDiceSet, vSortedSimulationResult: EYamsActions)
_nextPlayer ()
_onEndGame()
_resetChoices(choice: EChoices)
_hideGameBar(bHide: Boolean)
_showRules()
QAboutWidget
m_ui: Ui::QAboutWidget
QAboutWidget(parent: QWidget)
slot : showAboutQtWindow()
slot : showAboutOpenCVWindow()
QEndGameWidget
m_ui: Ui::QEndGameWidget
QEndGameWidget(sWinnerName: String, parent: QWidget)
getQuitButton(): QPushButton
getRestartButton(): QPushButton
CPlayerForm
m_HBoxLayout: QHBoxLayout
m_lblPlayerNum: QLabel
m_lptPlayerName: QLineEdit
CPlayerForm(parent: QYamsStartFormWidget, iPlayerNumber: Integer)
getInputName(): String
getQWidget(): QHBoxLayout
1..*
QYamsStartFormWidget
m_ui: Ui::QYamsStartFormWidget
m_ui2: CPlayerForm[1..*]
QYamsStartFormWidget(parent: QWidget)
signals : playerNameSetUp(String[1..*])
addPlayerNameInput()
onPlayerNameSetUp()
QPlayerGridsWidget
m_ui: Ui::QPlayerGridsWidget
m_mapPlayerGridViews: map<QPlayerGrid, CPlayerGridView>
QPlayerGridsWidget(parent: QWidget)
slot actualizeActionButtons(playerGrid: QPlayerGrid)
slot updateGrid(playerGrid: QPlayerGrid)
getAcesButton(): QPushButton
getTwosButton(): QPushButton
getThreeButton(): QPushButton
getFoursButton (): QPushButton
getFivesButton (): QPushButton
getSixesButton(): QPushButton
getBrelanButton(): QPushButton
getCarreButton(): QPushButton
getFullButton(): QPushButton
getSmallStraightButton(): QPushButton
getLargeStraightButton(): QPushButton
getYamsButton (): QPushButton
getSuperYamsButton (): QPushButton
getChanceButton(): QPushButton
addGrid(playerGrid: QPlayerGrid)
removeGrid(playerGrid: QPlayerGrid)
enableActionButtons(bEnable: Boolean)
CPlayerGridView
lblPlayerName: QLabel
lblAs: QLabel
lblTwos: QLabel
lblThrees: QLabel
lblFours: QLabel
lblFives: QLabel
lblSixes: QLabel
lblPreTotalSup: QLabel
lblBonus: QLabel
lblTotalSup: QLabel
lblBrelan: QLabel
lblCarre: QLabel
lblFull: QLabel
lblSmallStraight: QLabel
lblLargeStraight: QLabel
lblYams: QLabel
lblSuperYams: QLabel
lblChance: QLabel
lblTotalInf: QLabel
lblTotal: QLabel
CPlayerGridView(parameter: QPlayerGridsWidget, sPlayerName: String)
~CPlayerGridView()


Lancer de dés
Démarrage de la détection
Algorithme 1
Algorithme 2
[Si le nombre de dés est égal à 5]
[Si le nombre de points détectés par les deux algorithmes est égal]
Simulation et calcul des meilleurs choix
Propose les meilleurs choix à l'utilisateur
Attente de sélection par l'usager
Mise à jour de la grille de jeu
Passage au joueur suivant
[La grille est-elle terminée ?]
[Oui]
Étant donné qu'il n'est pas possible de passer son tour au Yams, si la grille du prochain joueur est finie, c'est aussi le cas pour tous les joueurs

5. Vision robotique

Pour la partie vision robotique, j'ai choisi d'implémenter l'analyse d'image sous forme de deux algorithmes devant retourner une information commune.
Si cette information est identique, on valide l'analyse de cette image.

Les deux algorithmes prennentune image passée en niveau de gris puis lissée (pour réduire le bruit). On réduit l'image à une taille de 640 x 480 afin de diminuer le nombre de pixels à traiter.

Afin de ne pas traiter d'image en mouvement, nous utilisons la fonction `cv::absdiff` qui permet d'obtenir une matrice contenant uniquement les pixels ayant changé entre l'ancienne et la nouvelle image.
On compte ensuite les pixels qui ne sont pas égaux à 0 (pas noirs) grâce à `cv::countNonZero`.
On établit ensuite un pourcentage. Si la similarité entre les deux images est inférieure à 40%, on ne fait pas de traitement pour celle-ci.

5.0.1 Algorithme 1 : Détection par aire minimum

La première étape consiste à appliquer le filtre de Canny sur l'image, puis à en extraire les contours avec `cv::findContours`.

Ceci fait, pour chacun de ces contours, on cherche le rectangle d'aire minimale entourant jeu jeu de points formant le contour.
Si l'aire minimale de ce rectangle est supérieure ou égale à l'aire minimum (fixée) d'un dé et inférieure à la valeur maximum (fixée aussi) d'un dé, on le compte comme étant le rectangle contour d'un dé.

Ensuite, pour chaque rectangle estimé être un dé, on extrait la portion de l'image lui correspondant puis on recommence comme ci-dessus en comparant cette fois les contours enfant du dé.
L'aire des rectangles trouvés doivent ici être compris entre une aire minimum et une aire maximum, toutes deux fixées, pour un point de dé.
On retourne ici pour chaque dé, son rectangle parent, ainsi que ses rectangles enfants (les points), puis le nombre de point total détecté.

5.0.2 Algorithme 2 : Détection par blob / goutte

Nous utilisons ici le `cv::SimpleBlobDetector` d'OpenCV, paramétrable avec `cv::SimpleBlobDetector::Params`.
Cet outil permet de détecter des groupes de pixels connectés qui partagent une même propriété au sein d'une image (ici les points des dés).
On récupère ainsi des *keypoints*.
On retourne donc le nombre de *keypoints* détectés qui correspond aux nombres de points total des dés.

Si le nombre de points détectés de part et d'autre sont égaux, on considère que l'analyse est correcte.
On transmet le résultat à l'utilisateur uniquement si l'exécution sur deux images différentes et successives donnent le même résultat afin de réduire les risques d'erreur.

Affin d'accélèrer le porcessus, on exécute les deux algorithmes dans deux *thread* différents si cela est possible. Cela se vérifie grâce à la fonction `std::thread::hardware_concurrency` de la bibliothèque standard du C++ qui retourne le nombre de *thread* qu'il est possible de lancer sur la machine.
Ce nombre doit être supérieur ou égal à deux.

6. Annexe

6.1 Version des logiciels et bibliothèques utilisés (en mai 2020)

- Windows 10 build 1909 (Septembre 2019)
- Visual Studio 2019 (16) et le CMake qui lui est intégré
- OpenCV 4.3.0 (2020-04-06)
- Qt 5.12.6

Bibliographie

[1] Qt - www.qt.io.

[2] Documentation Qt - doc.qt.io.

[3] OpenCV - opencv.org.

[4] Documentation OpenCV - docs.opencv.org.

[5] Dépôt GitHub du projet Yams - github.com/corentin703/CEGEP-Projet-Yams.