HOW TO MAKE A GAME SIMULATION

Presented by: Jason Wang on behalf of Marianopolis Robotics

1.0 Why Game Simulations are Beneficial

Due to the nature of this year's competition, a lot of the game depends on strategy. Deciding when to activate the power-ups and whether to prioritize some actuators over others is crucial to a team's performance during the competition. While it is simpler to figure out what your robot has to do to score the most amount of points, thinking about what your teammate and opponents might be doing is way more complex. This is where game simulations come into play. They allow us to control the actions of all the game players and practice different strategies to see which ones give the best outcome.

1.1 What Led to us Needing a Game Simulation

During our first couple meetings we spent a lot of time debating potential game strategies. This was important because the angle we would choose to take would affect how we built the robot (ex. prioritizing speed). By the end of the meeting, we agreed that the robot being able to move around the field quickly is crucial. However, we were not able to decide on a proper strategy in terms of when to hit which actuators. This led to us deciding that we should make a game simulation to test out our different strategies against opponents with varying strategies as well.

2.0 Researching for the Game Simulation

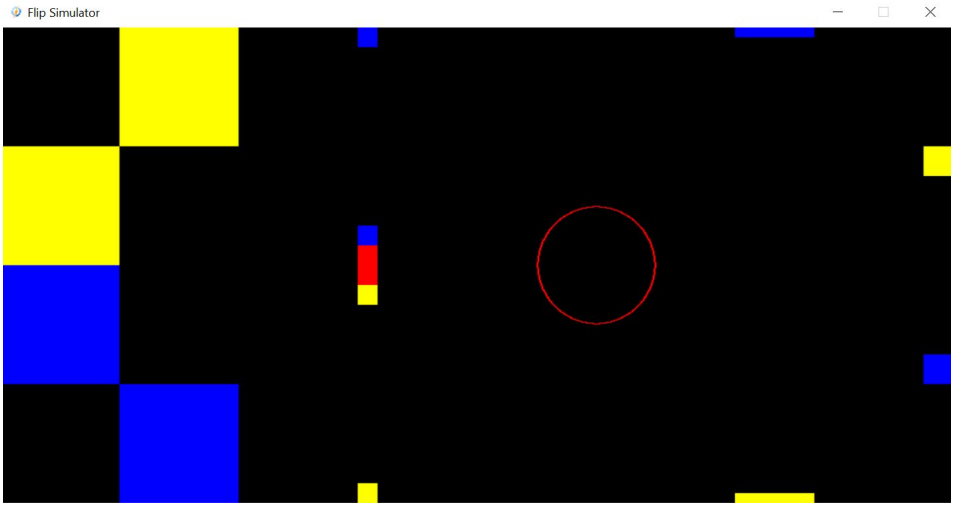
Due to the nature of this year's competition, replicating the fundamental logic of the game into a video game is not only practical, but also simple. At its core, the competition is a

matter of driving a robot that goes around and presses a variety of buttons in a strategic manner. So translating it into an interactive simulation, the physical challenges of pressing the real buttons can be simplified to buttons that have a delay when pressed, which renders the task of making the simulation quite straightforward.

Normally when creating a game, it is much better to work with multiple layers and sprites as it allows for better organization. But since the game is just one field with a total of 12 buttons to be pressed, the game can be made by directly manipulating pixels since there is not a confusing amount of information to keep track of. This also requires a less advanced level of programming knowledge, which in turn makes creating a simulation possible to everyone.

2.1 Making the Game Simulation

Most languages are similarly good at creating 2D games. I chose Python(version 3.7) and used the pygame(version 1.9) module as it is more suitable for introductory level programmers.



First a field must be created with all the appropriate markings must be made. Any resolution size works, so long as the proportions are proportional to the actual measurements of the

field. Each section of the field marking either the starting positions, boundaries, or powerups must be made by designating its pixel boundaries. This in turn lets you list the relevant locations for when boundaries must be programmed.

```
'''Starting Sections'''
yellow1 = pygame.Rect(120, 0, 120, 120)
yellow2 = pygame.Rect(0, 120, 120, 120)
blue1 = pygame.Rect(0, 240, 120, 120)
blue2 = pygame.Rect(120, 360, 120, 120)
'''Arch Sections'''
yArch1 = pygame.Rect(360, 260, 20, 20)
yArch2 = pygame.Rect(360, 460, 20, 20)
bArch1 = pygame.Rect(360, 0, 20, 20)
bArch2 = pygame.Rect(360, 200, 20, 20)
bridge = pygame.Rect(360, 220, 20, 40)
'''Power Buttons'''
ySwitch = pygame.Rect(930, 120, 30, 30)
bSwitch = pygame.Rect(930, 330, 30, 30)
yMshrm = pygame.Rect(740, 470, 80, 10)
bMshrm = pygame.Rect(740, 0, 80, 10)
```

Section Boundaries on a 960x480 resolution

Once the field is made, a sprite needs to be created. The sprite needs to have an updatable x-y coordinate which can changed to move it around. Afterwards the boundaries of where the sprite can move must be explicitly stated to emulate the walls of the field and other obstacles. Make this functional for 4 sprites.

```
if 0 < playerX1 + playerX1change < 896:
    if playerY1 < 20:
        if (playerX1 + playerX1change > 296) and (playerX1 + playerX1change < 380):
            print("blocked")
        else:
            playerX1 += playerX1change
    elif 136 < playerY1 < 280:
        if (playerX1 + playerX1change > 296) and (playerX1 + playerX1change < 380):
            print("blocked")
            #print(playerX1, playerY1)
        else:
            playerX1 += playerX1change
    elif playerY1 > 396:
        if 296 < playerX1 + playerX1change < 380:
            print("blocked")
        else:
            playerX1 += playerX1change
    else:
        playerX1 += playerX1change
```

x-coordinate boundaries of a moving sprite

Once a field and a moving sprite with set boundaries is created, a score tracking system needs to be created. Since the CRC team has generously shared a simulation that emulates the scoring system of the competition, this remains optional. But if someone wishes to make one, it is essentially just a lot of data manipulation with timers. The inputs for the buttons just need to be based off of x-y coordinates of the sprites, and the output indicating the availability of power ups during a heat can be represented either through a separate dialog box, or through the colour of the field itself.

Comment créer une simulation de jeu
Présenté par: Jason Wang et Marianopolis Robotics

1.0 Pourquoi les simulations de jeu sont bénéfiques

À cause de la nature de la compétition de cette année, une grande partie du jeu dépend de la stratégie. Décider quand activer les bonus et s'il faut prioriser certains actuateurs est crucial pour la performance d'une équipe pendant la compétition. Bien qu'il soit plus simple de comprendre ce que votre robot doit faire pour marquer le plus de points, penser à ce que votre coéquipier ou vos adversaires pourraient faire est beaucoup plus complexe. C'est là que les simulations de jeu entrent en jeu. Ils nous permettent de contrôler les actions de tous les joueurs et de pratiquer différentes stratégies pour voir lesquelles donnent le meilleur résultat.

1.1 Ce qui nous a amenés à avoir besoin d'une simulation de jeu

temps à débattre des stratégies de jeu potentielles. C'était important parce que l'angle que nous choisissions de prendre affecterait la façon dont nous construisions le robot (ex. prioriser la vitesse). À la fin, nous avons décidé qu'il est crucial que le robot peu se déplacer rapidement sur le terrain. Cependant, nous n'avons pas été capables de décider d'une stratégie appropriée en termes de quand frapper les actuateurs. Cela nous a amenés à décider que nous devrions faire une simulation de jeu pour tester nos différentes stratégies contre des adversaires aux stratégies différentes.

2.0 La Recherche pour la Simulation de Jeu

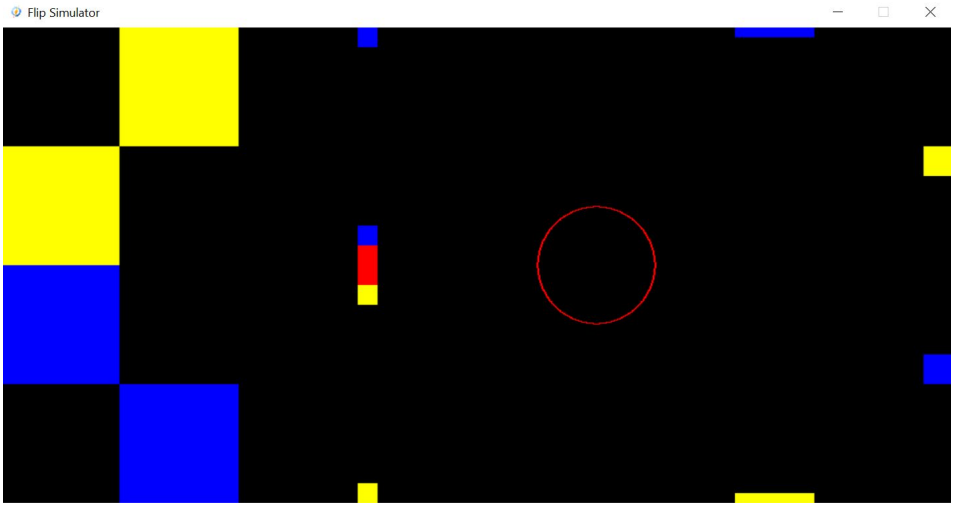
En raison de la nature de la compétition de cette année, le reproduction de la logique fondamentale du jeu dans un jeu vidéo est non seulement pratique, mais aussi simple. Fondamentalement, la compétition est basée sur la conduite d’un robot qui actionne des boutons d’une manière stratégique.Donc, en traduisant cela

dans une simulation interactive, les défis physiques d'actionner les boutons peuvent être simplifié à des boutons à retardement, qui rend la simulation très intuitive à créer.

Normalement lors de la création d'un jeu, il est mieux de travailler avec plusieurs couches puisque cela rend le projet plus organisé. CEpendant, comme le jeu n'est qu'un terrain avec 12 boutons à être actionnés, le jeu peut être fait directement par la manipulation des pixels puisqu'il n'y a pas une grande quantité d'information à gérer. Ceci demande aussi moins en terme de connaissances en programmation, qui rend la simulation possible pour tous.

2.1 La création de la simulation de jeu

La plupart des langues sont également bonnes pour la création des jeux 2D.J'ai choisi Python (version 3.7) et j'ai utilisé le module pygame (version 1.9) puisqu'ils sont plus accessibles pour les programmeurs de niveau débutant.



Premièrement, un terrain doit être créée avec tout l'étiquetage nécessaire. La résolution n'importe pas, du moins que les proportions sont équivalentes aux proportions du terrain réel. Chaque section du terrain qui désigne soit les positions de départ, les limites ou les bonus voient être fait par

manipulation des pixels. Ceci vous permet de lister toutes les locations importantes des limites qui doivent être programmées.

```
'''Starting Sections'''
yellow1 = pygame.Rect(120, 0, 120, 120)
yellow2 = pygame.Rect(0, 120, 120, 120)
blue1 = pygame.Rect(0, 240, 120, 120)
blue2 = pygame.Rect(120, 360, 120, 120)
'''Arch Sections'''
yArch1 = pygame.Rect(360, 260, 20, 20)
yArch2 = pygame.Rect(360, 460, 20, 20)
bArch1 = pygame.Rect(360, 0, 20, 20)
bArch2 = pygame.Rect(360, 200, 20, 20)
bridge = pygame.Rect(360, 220, 20, 40)
'''Power Buttons'''
ySwitch = pygame.Rect(930, 120, 30, 30)
bSwitch = pygame.Rect(930, 330, 30, 30)
yMshrm = pygame.Rect(740, 470, 80, 10)
bMshrm = pygame.Rect(740, 0, 80, 10)
```

Limites des sections sur une résolution de 960x480

Quand le terrain est fait, un sprite doit être créée. Le sprite doit avoir des coordonnées x-y qui modifiables sur le volant afin que le sprite bouge. Après, les limites du mouvement du sprite doivent être dictés explicitement pour simuler les murs du terrain et des obstacles. Rendez cela fonctionnel pour 4 sprites.

```
if 0 < playerX1 + playerX1change < 896:
    if playerY1 < 20:
        if (playerX1 + playerX1change > 296) and (playerX1 + playerX1change < 380):
            print("blocked")
        else:
            playerX1 += playerX1change
    elif 136 < playerY1 < 280:
        if (playerX1 + playerX1change > 296) and (playerX1 + playerX1change < 380):
            print("blocked")
            #print(playerX1, playerY1)
        else:
            playerX1 += playerX1change
    elif playerY1 > 396:
        if 296 < playerX1 + playerX1change < 380:
            print("blocked")
        else:
            playerX1 += playerX1change
    else:
        playerX1 += playerX1change
```

Coordonnées x d’un sprite en mouvement

Une fois que le terrain et le sprite mobile avec des limites fixes sont créés, un système pour garder le pointage doit être créer. Puisque l’équipe CRC a partagé généreusement une

simulation de leur système de pointage, cette étape est facultative. Cependant, si quelqu'un voudrait le faire, c'est essentiellement beaucoup de manipulation des chiffres et des minuteries. Les entrées des boutons doivent être basées sur les coordonnées x et y des sprites et les sorties doivent indiquer les bonus disponibles durant la compétition peuvent être représentés soit par une boîte à dialogue ou par la couleur du terrain lui-même.