GNG1503

Manuel d'Utilisateur du Projet de Conception

[Prototype final]

Soumis par:

[FA-18]

[Issiaka Soumaoro,300101398]

[Adam Taktek, 300110268]

[Mariam Farota,300087955]

Date: 04/12/2019

Université d'Ottawa

Résumé

Notre jeu de tracteur est un jeu vidéo 2D simple, accessible à très bas coût (99$ et un(e) moniteur/ Télé). Il immerge le joueur dans un jeu captivant pendant une minute complète. À la fin l'utilisateur voudra avec enthousiasme partager l'expérience captivante au prochain visiteur.

Le jeu consiste à déplacer son tracteur dans une carte de champs en 2D pour ramasser le plus de blé sans jamais sortir du champs ou percuté le blé qui traîne derrière le tracteur.

Table des matières

1 Introduction

1.1 Présentation du problème

Un client, le musée d'agriculture du Canada nous a demandé de travailler sur un projet de conception d'un produit à installer dans un de ses bâtiments. Depuis 2002, il y a sur le site du musée une exposition de tracteur dont un Interactif qui a été très populaire. Malheureusement en 2019, le simulateur de tracteur n'est plus en fonction. Le Musée a alors pris contact avec nous et a pu s'informer qu'il y avait une opportunité avec l'université d'Ottawa afin de rénover le simulateur.
Nous sommes alors chargé par le musée (client) de mettre à jour la cabine de tracteur pour offrir une expérience idéale d'un tour de tracteur aux visiteurs (utilisateurs). Voici notre énoncé de problème produite selon les besoins des clients:

Concevoir un tracteur interactif, moderne, autonome, accessible, sécuritaire, robuste et à coût abordable pour les investisseurs qui pourra donner l'expérience idéale d'une simulation d'un tour de tracteur aux visiteurs.

L'**importance** de répondre à ce problème est que le musée puisse donner aux visiteurs une expérience divertissante et mémorable dans la cabine de tracteur qui pourra remplacer l'ancien interactif de tracteur. Les motivateurs du besoin du produit d'un point de vue économique sont l'amélioration du revenue du client et de l'état émotionnel des visiteurs. Premièrement, ce produit doit améliorer l'état émotionnel des utilisateurs, et implicitement en satisfaisant les visiteurs il pourra attirer plus de visiteurs au Musée.

Les **besoins fondamentaux** de notre client sont que le produit soit interactif, moderne, autonome, accessible, sécuritaire, robuste et à coût abordable. Ceci est important pour le client.

1.2 Notre prototype

On a ainsi créer un jeu qui offre une interaction accessible (français ou anglais, passive ou active). Notre prototype donne l'accès à un jeu vidéo à l'aide d'un micro-ordinateur peu coûteux.

Ce qui fait que notre produit est le meilleur est l'accent sur l'agriculture de précision, le bas prix comparé aux autres produit des collègues, qu'il est fonctionnelle et est robuste.

- Il coûte 99$ et un écran de votre choix avec un câble VGA. Beaucoup de nos collègues ont dépassé le budget de 100$ et ont besoins d'un ordinateur (MAC/UNIX/WINDOWS) qui implique un investissement supplémentaire de plus de 300$ (pour un prix total supérieur à 400$). Le notre est seulement 100 $ et un écran.
- Le logiciel et le jeu fonctionne. On peut y jouer déjà. Certains de nos collègues ont un prototype que ne fonctionne pas encore.
- Le produit est solide. Notre produit utilise des boutons qui sont durable. On n'utilise pas de potentiomètre qui se brise facilement et qu'il faudra constamment remplacer.
- De plus, notre prototyp met un accent sur l'agriculture de précision: On demande à l'utilisateur de déplacer son tracteur sur une carte GPS du champs où les semances sont prêtes pour la récolte.

2 Comment le prototype est construit

2.1 Catégorie

2.2 LDM (Liste des Matériaux)

Numéros	Descriptif	Prix à l'unité	Quantité	Fournisseur	Prix d'achat
1	6 boutons	0.70$	6	Makerstore	4.2
2	Raspberry Pie Zero (KIT)	36 $	1	Amazon	36
3	Micro HDMI to VGA	**14,95 CDN$**	1	Amazon	15
4	Micro SDHC card 32GB	10.40 $	1	Amazon	10.40
5	Micro OTG to Multiple USB ins	10,72 $	1	Amazon	11
6	-Male header connector -Filament PLA (imprimante 3D)	0$	1	Makerstore	Offert
7	Haut parleur avec entrée AUX	0$	1	Nous	0
8	Plaques de MDF	2.5$	6	Makerstore	15$
9	(Taxes)	7.84	1	Amazon	7.84
Total					99.44$

Voici quelques indices pour une amélioration du produit

- Remplacer le *raspberry pi zero w* & *Male header connector* & *Micro HDMI to VGA* & *Micro SDHC card 32GB*
  → par **Raspberry Pi 3 Model A+ Starter Kit** sur canakit.com pour environ 50 $
  (cela revient mieux économiquement et en terme de performance)
- Remplacer les 6 boutons du Maker Store par des boutons type arcade (lien) sur Amazon pour environ 16 $.
  (Des gros boutons seront beaucoupe plus accessible pour tout le monde; modifier en conséquence les trous sur de la plaque avant)
- Le choix du MDF est plutôt arbitraire. On fait ce choix dû à son accessibilité et son bas prix. D'autres matériaux sont envisageable (l'acier, l'acrylique).

2.3 Liste d'équipements

-Logiciels: solidwork, inkscape, IDE de python, éditeur de vidéo .

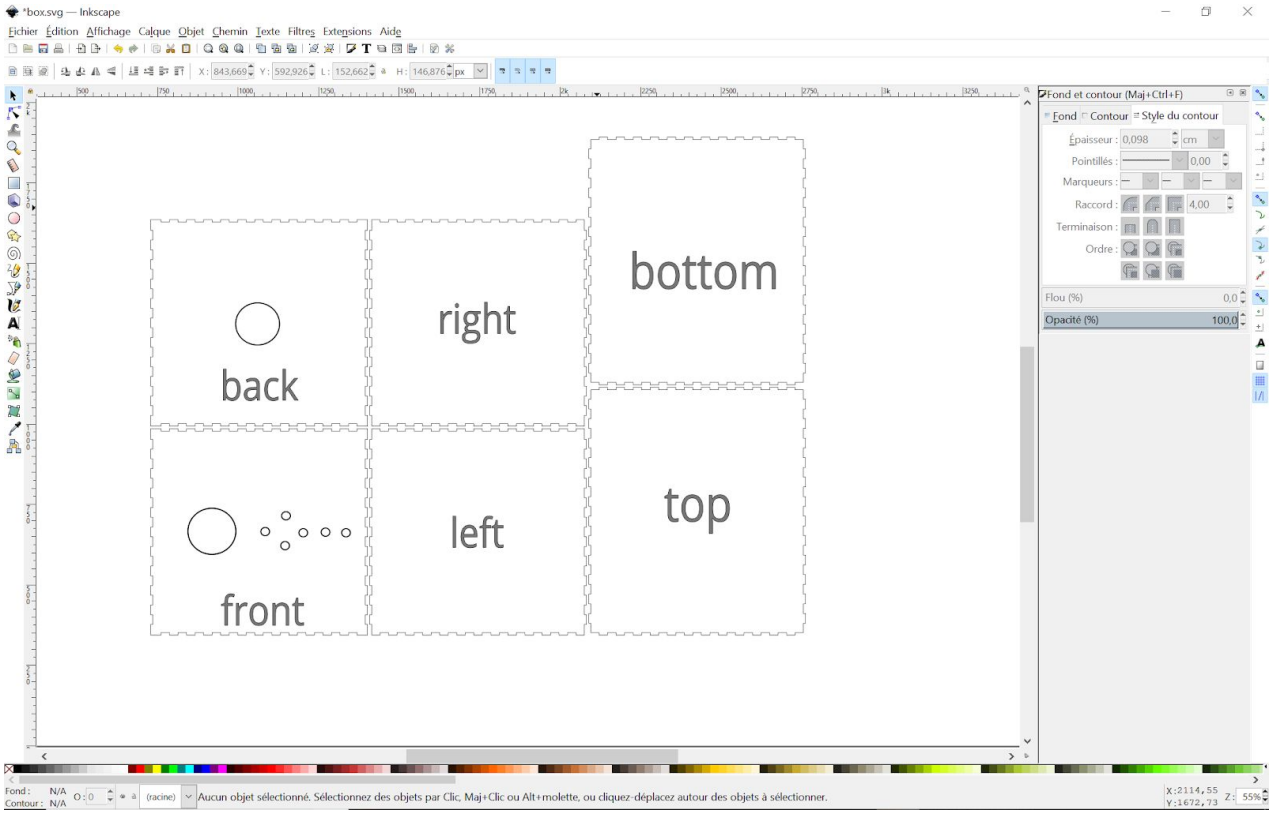
-Outils: ordinateur(s), Imprimante 3D, machine à découpe laser , matériels pour faire des soudures, fusil à colle chaude.

2.4 Instructions

-Domaines mécanique et physique:

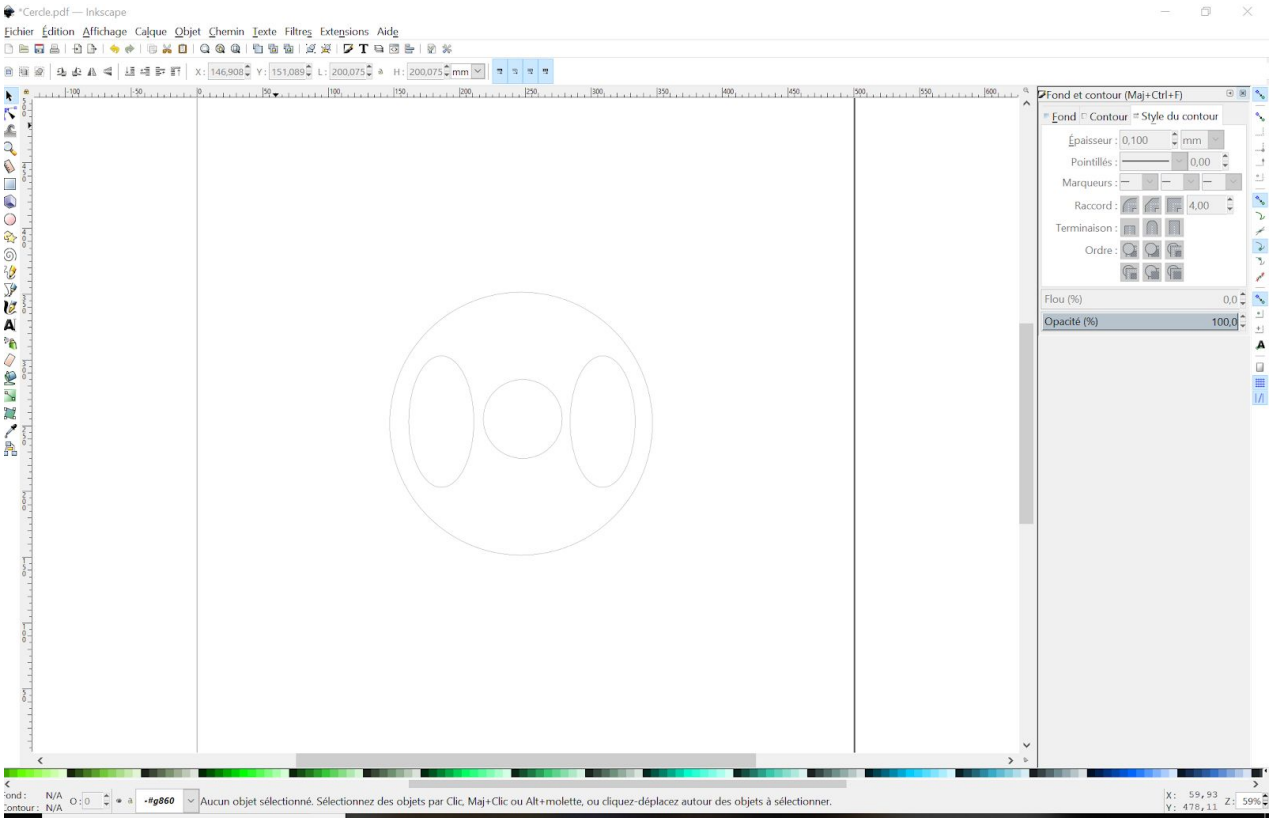
- **La Boite:**

La boite a été faite en pièces détachées. On a utilisé le logiciel Inkscape pour découper du MDF au laser. On a réalisé les modèles de chaque côté de la boite en prenant soin de mettre les troues pour le volant, les boutons, pour laisser passer le fil d'alimentation etc.



- **volant:**

Pour le volant, de même que pour la boite, on a utilisé un logiciel Inkscape pour faire le modèle du volant ensuite nous l'avons découpé dans du mdf au laser, puis finalement on l'a peint.



- **Cylindre pour le volant:**

Ensuite, on a utilisé le logiciel solidwork pour créer un cylindre avec les dimensions adéquates .Ce cylindre sert comme fixeur pour maintenir le volant à la boite(tableau de bord) et qu'on puisse tourner ou manier le volant dans la direction qu'on veut.

- **Les pédales:**
    - **Partie support des pédales:**

Matériaux requis : 2 cartons rectangulaire
Avant la construction des pédales elle-même, on doit faire en sorte qu'elles restent ensemble, et tiennent debout. On construit en cartons un support pour les pédales.
Photo 1: Allure de la construction:

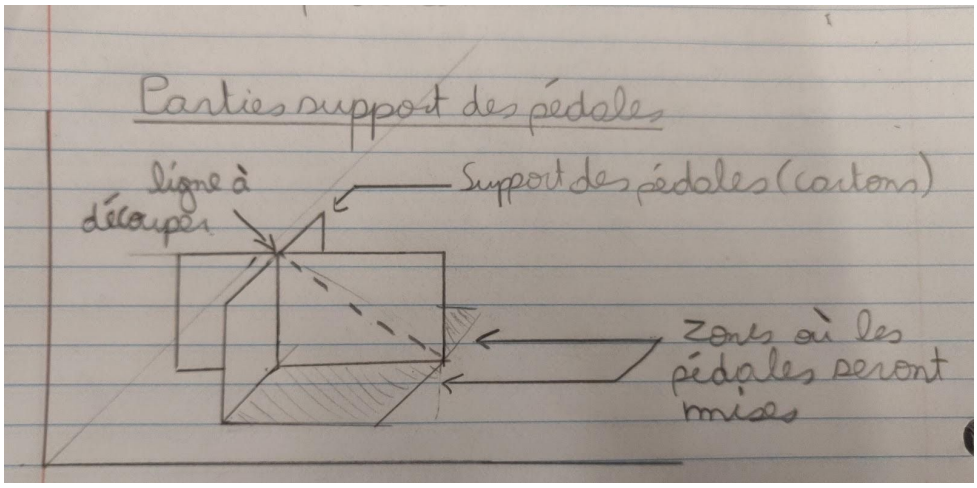
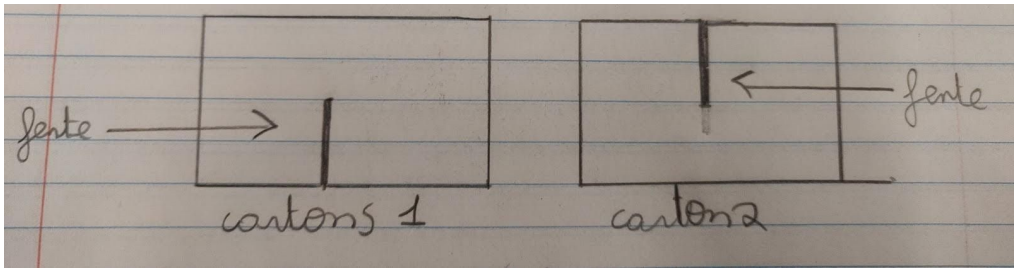


Photo 2 : Spécifications et détails des matériaux de la construction:



- **Partie - Les pédales elle même:**

Matériaux requis : 2 autres cartons rectangulaire très long.

Maintenant, on utilise 2 autres cartons pour faire les pédales.
Un carton (1) sera celui sur lequel l'utilisateur posera son pied.
L'autre carton (2) servira d'une sorte de ressort pour que la pédale ne reste pas simplement sur le sol. De cette manière l'utilisateur aura l'expérience d'avoir une pédale qui remonte debout lorsqu'il ne pose pas son pied dessus.

Photo 1: Allure de la construction:

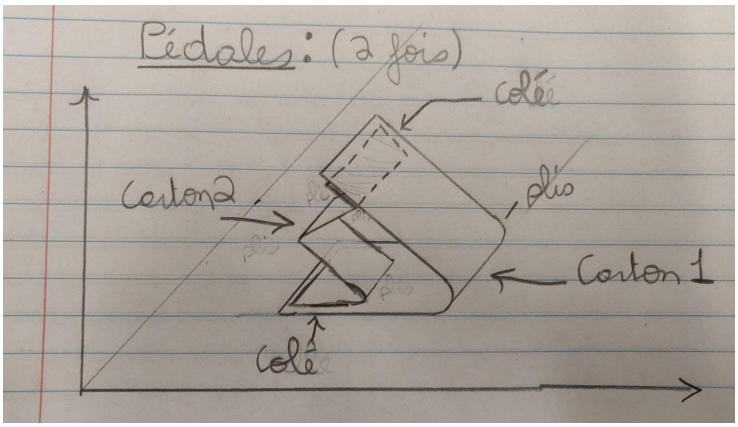
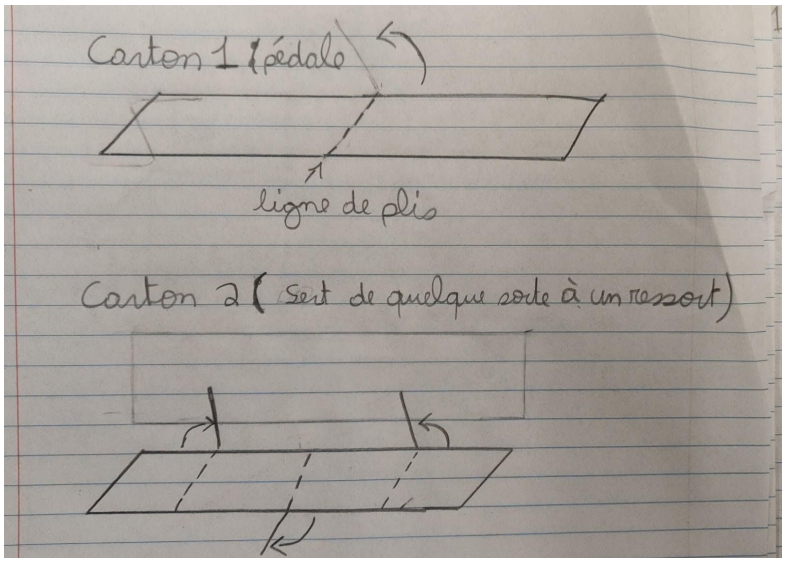


Photo 2 : Spécifications et détails des matériaux de la construction:
Pour deux pédales, on fait deux fois ce qui suit sur les images.


Carton 1 (pédale
ligne de plis
Carton 2 (sert de quelque sorte à un ressort)

-Domaine de l’Électrique et l'électronique:

- **Câbles:**

  Concernant l’électronique, nous avons tout ce qui est présenté dans l’image ci dessous

1: Alimentation principale pour faire démarrer le tout;

2: Un relai d’alimentation auquel est relié le haut parleur(3) et auquel sera relié les autres composantes (souris, etc.);

3: Haut parleur à l’alimentation et aussi au cable HDMI;

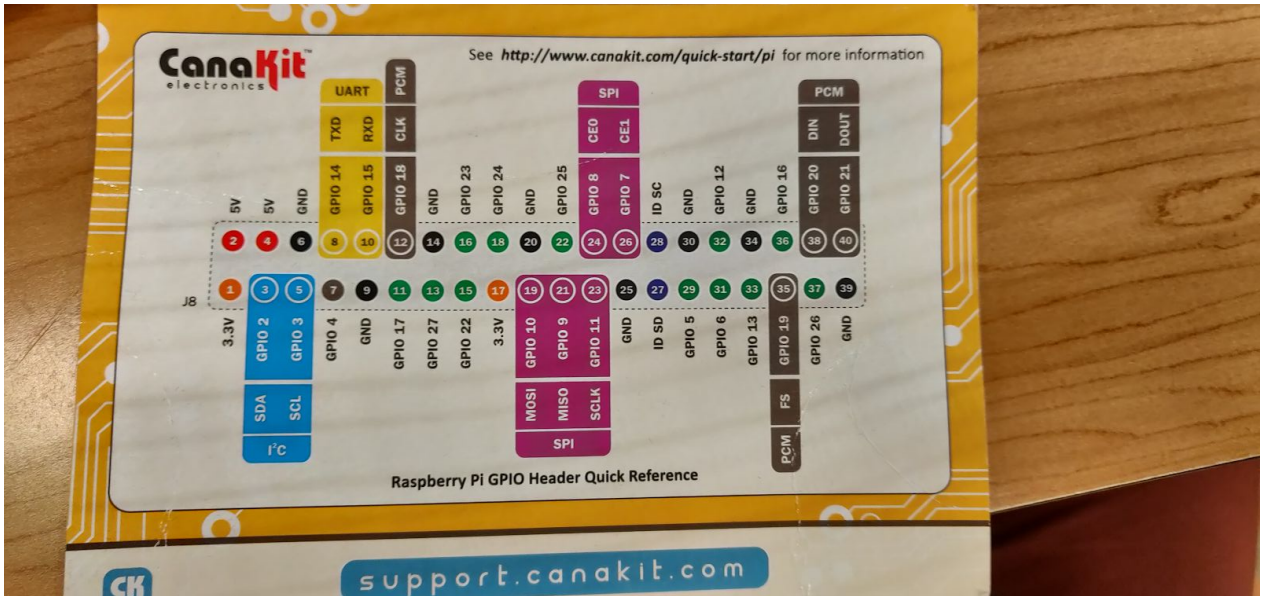
4: Cable HDMI porteur des branchements audio (AUX) et visuel (VGA).

- **Le contrôleur - Raspberry pie Zero W:**

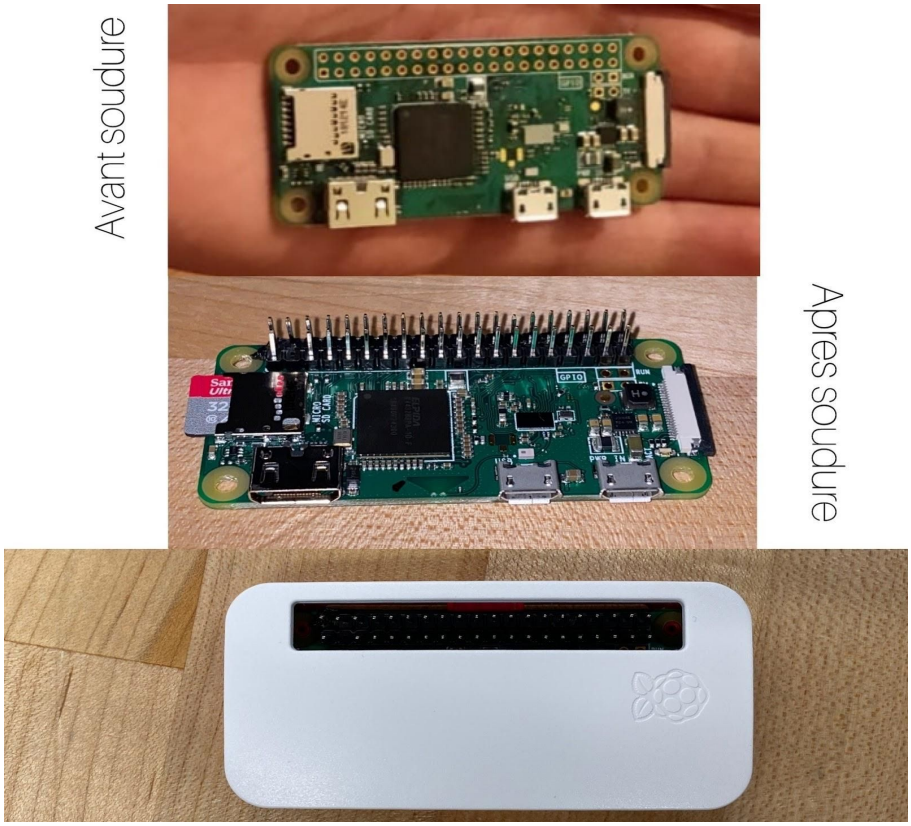
Pour notre produit, un contrôleur est élémentaire pour pouvoir analyser les entrées d'informations venant de l'utilisateur, et puis pour pouvoir créer des rétroactions qui seront envoyé à l'utilisateur de manière visuel et auditive.

Nous avons reçus le raspberry pie zéro W. Nous avons pu tester les sorties d'informations (les sons et données visuelles sont émise et peuvent être perçu). Nous devons maintenant faire en sorte qu'il est possible de créer des entrées d'informations.

Pour ce faire, nous savons qu'il nous faut souder un "male header connector" pour pouvoir brancher des composants aux raspberry pie. On consulte la documentation des pins GPIO ci-après pour repérer les pins essentiel aux dispositifs.



On estime que les pins importants sont les GPIO 17, 18 , 27, 22, 23 et 24 (pour connecter 6 boutons) , des pins d'alimentation 2 de 3.3V ,1 de 5V , et 3 GND. C'est ceux -ci qu'on soude:



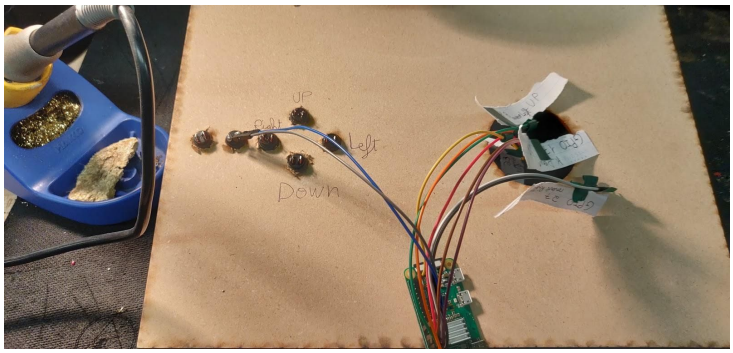
Sur l’image ci dessus c’est notre raspberry pie.

Il nous reste plus qu’à assembler des boutons pour donner vie à l'interaction.

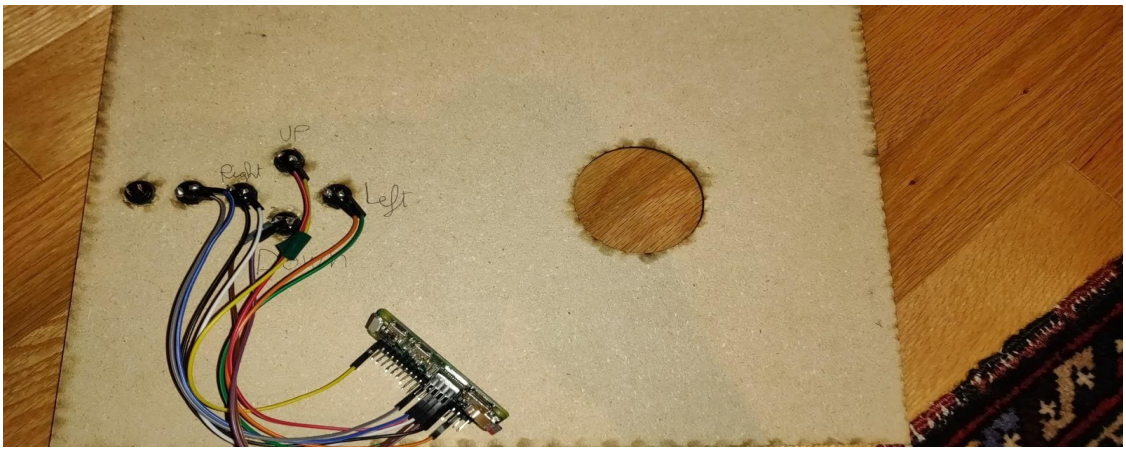
- **Boutons:**

On a choisi 6 boutons qui sont les touches :start, power, up, down, left et right.À chaque boutons on soudés deux fils (jumper et wier) et directement liées aux GPIO pins du raspberry pie.

Les boutons sur la plaque



1er soudage



dernier soudage

-Logiciel:

- **Codages(python):**

Enfin pour notre simulateur nous l'avons codé en python. Ce code permet d'afficher en sortie des informations visuel et auditive qui sont transmis à un écran et un haut parleur. Il permet aussi d'analyser les informations provenant des boutons et de faire les mouvements désiré dans le jeu.

Pour faire fonctionner ce code il faut mettre dans un même fichier, le code python3 "Tracteur.py" et la "vidéotracteur.mp4". Vous pouvez trouver le code dans *l'Appendix I* ou prochainement sur MakerRepo avec la vidéo.

Le code python pourrait subir quelques amélioration si jamais il faisait réellement partie d'une exposition. Par exemple, il serait bénéfique que le jeu se réinitialise récursivement pour que plusieurs utilisateurs puisse jouer à la suite, jusqu'à ce qu'un employé décide de fermer le jeu. Il faudrait maîtriser les appel récursif de fonction et des boucle infini en python pour le faire. Ça été difficile pour nous de le réaliser du au manque de temps malgré nos plusieurs essais de le faire.

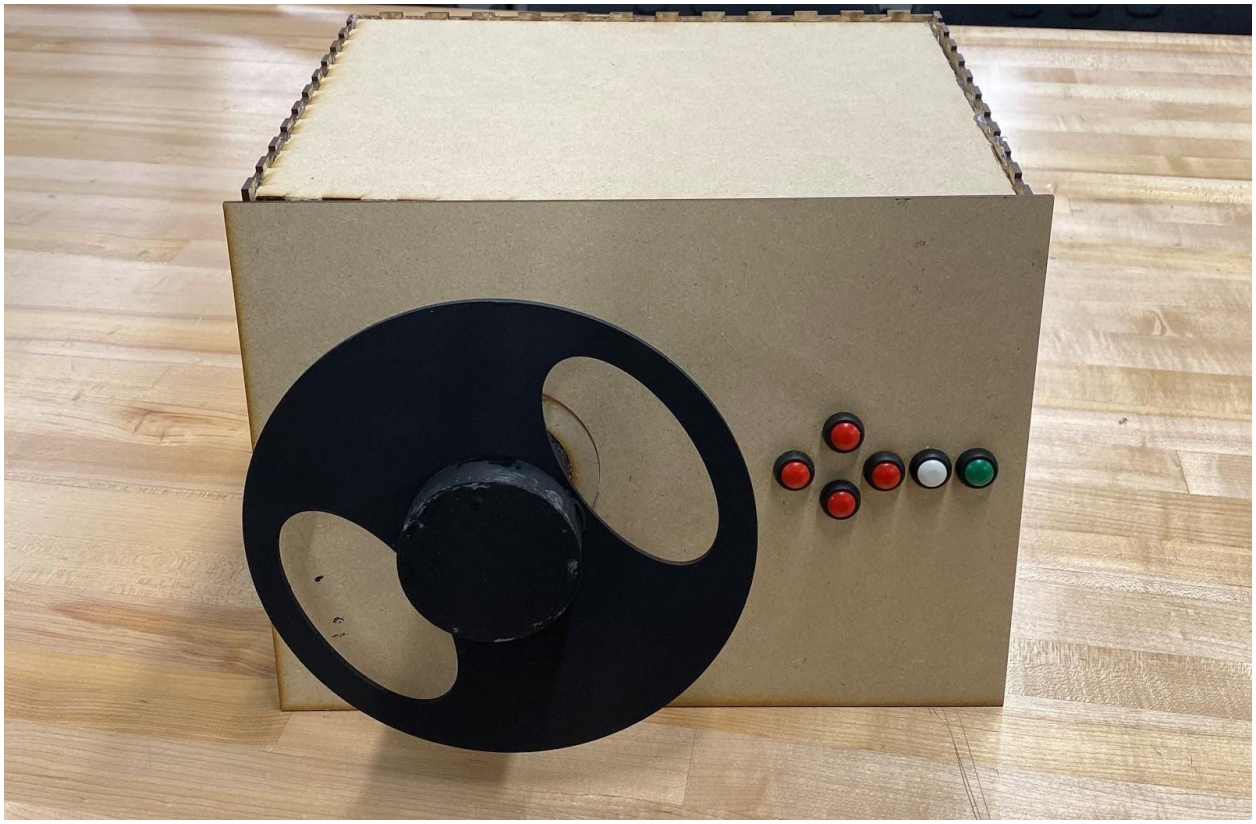


Image du prototype final (plus un écran)

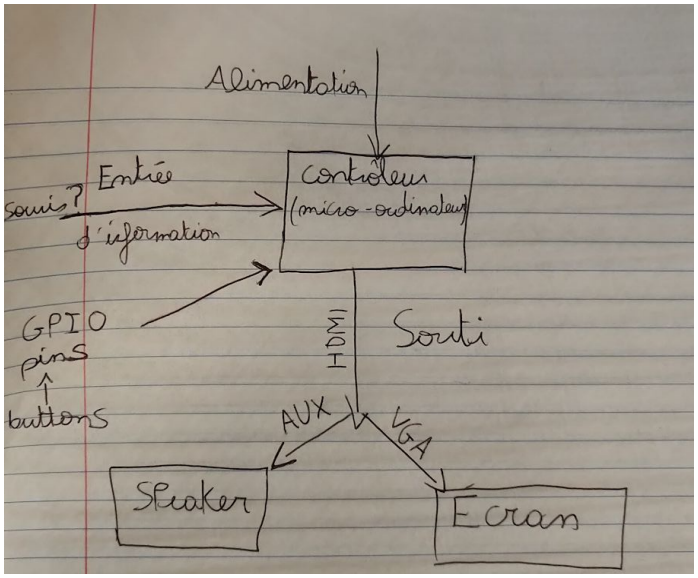
3 Comment utiliser le prototype

Pour utiliser le prototype, il faut tout d'abords trouver un écran (avec un câble VGA) et une souris auquel connecter le raspberry pie. Ensuite brancher le câble d'alimentation du raspberry pie dans une prise (on peut attendre que la LED sur le raspberry pie fini de clignoter). On va voire le bureau de l'ordinateur s'afficher (le raspberry pie fonctionne avec un système d'exploitation Unix) on peut naviguer au fichier "video player", puis dans le fichier "game". Cliquer droit sur le fichier "Tracteur.py" et encore cliquer sur ouvrir avec "geany's editor". Ensuite dans l'application qui s'ouvre cliquer sur "Run" et le simulateur démarre!

Pour utiliser le prototype en sécurité, il faudrait éviter de jouer avec les câbles et il faut utiliser le prototype avec délicatesse. Il ne faut pas tenter d'endommager le prototype ou le frappé violemment. Il ne faut pas soumettre le prototype à des température qui empêche le fonctionnement d'un ordinateur que ce soit un température chaude ou froide. Il faut aussi respecter les marges d'humidité pour le fonctionnement d'un ordinateur.

Le fonctionnement du prototype est simple. Lorsque le code est joué il suffit d'appuyer sur les boutons pour interagir avec le prototype. Lorsqu'un bouton est pesée cela permet le passage d'un courant jusqu'au pin GPIO du raspberry pie. À travers le code on peut interpréter ce passage de courant pour que l'utilisateur puisse interagir avec le contenu de l'écran.

Pour résumer le fonctionnement du prototype, il nécessaire d'avoir un alimentation de courant à l'ordinateur. Aussi, il y a des informations entrantes dans le système à travers la souris et les boutons. Puis l'ordinateur envois des information à l'écran et le haut parleur. Voici ci -dessous un diagramme synoptique qui résume le fonctionnement de notre prototype.



4 Comment maintenir le prototype

Pour faire fonctionner le prototype, il faudrait des températures comprises entre 2 C et 27 C. S'il faut qu'il faut qu'il fonctionne à des températures hors de cette intervalles, il est possible d'installer un ventilateur ou une résistance à l'intérieur de la boîte.

La boîte peut supporter quelques mouvements. Mais elle risque de casser si on donne intentionnellement des coûts violents dedans.
L'électronique devrait résister pour longtemps, l'ordinateur aussi. On estime leur duré de vis à 10 ans.
Pour ce qui est des boutons il se peut qu'il faille les remplacer après quelques années dû à l'utilisation intensives des visites. Pour le faire, il faut changer un bouton à la fois pour ne pas meler les fils.

5 Conclusions et recommandations pour les travaux futurs

Lors de ce travail on a pu apprendre certaines compétences tels que la gestion de conflits, la gestion de temps et de projets, donner et recevoir de la rétroaction ainsi que l'expérience utilisateur et aussi une présentation efficace.

Ce prototype est le meilleur.

On choisis le MDF parmis les autres matériaux par ce que le coût est abordable et il est facile à utiliser, on peut envisager d'autres matériaux. Quelqu'un pourrait se charger d'améliorer le code python pour faire les amélioration suggéré.

6 Bibliographie

https://www.sound-fishing.net/sons/tracteur

https://www.raspberrypi.org/forums/

https://docs.python.org/3/

APPENDICES

APPENDICE I: Fichiers de conception

Inclure tous les fichiers de conception dans MakerRepo avec des explications ici pour que les clients ou autres étudiants de semestres suivants puissent continuer votre projet. Aussi fournir le lien MakerRepo pour votre projet.

MakerRepo

```
import random, pygame, sys
from pygame.locals import *
from time import sleep
import os
import subprocess, os
import RPi.GPIO as GPIO


GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
GPIO.setup(18, GPIO.IN, pull_up_down=GPIO.PUD_UP)
GPIO.setup(17, GPIO.IN, pull_up_down=GPIO.PUD_UP)
GPIO.setup(23, GPIO.IN, pull_up_down=GPIO.PUD_UP)
GPIO.setup(27, GPIO.IN, pull_up_down=GPIO.PUD_UP)
GPIO.setup(22, GPIO.IN, pull_up_down=GPIO.PUD_UP)

FPS =9
WINDOWWIDTH = 1240#1240
WINDOWHEIGHT = 480 #480
```

```
CELLSIZE = 40
assert WINDOWWIDTH % CELLSIZE == 0, "Window width must be a multiple of cell size."
assert WINDOWHEIGHT % CELLSIZE == 0, "Window height must be a multiple of cell
size."
CELLWIDTH = int(WINDOWWIDTH / CELLSIZE)
CELLHEIGHT = int(WINDOWHEIGHT / CELLSIZE)

#            R    G    B
WHITE     = (255, 255, 255)
BLACK     = (  0,   0,   0)
RED       = (255,   0,   0)
GREEN     = (  0, 255,   0)
DARKGREEN = (  0, 155,   0)
DARKGRAY  = ( 40,  40,  40)
BEIGE = (225, 225, 200)
YELLOW= (255,255,102)
BGCOLOR = BLACK

UP = 'up'
DOWN = 'down'
LEFT = 'left'
RIGHT = 'right'

HEAD = 0 # syntactic sugar: index of the worm's head

def main():
    while True:
        partieun()


def partieun():
    global FPSCLOCK, DISPLAYSURF, BASICFONT
    pygame.init()
    FPSCLOCK = pygame.time.Clock()
    DISPLAYSURF = pygame.display.set_mode((WINDOWWIDTH, WINDOWHEIGHT),
0,12)
    BASICFONT = pygame.font.Font('freesansbold.ttf', 20)
    pygame.display.set_caption('GPS- Analyse du champs')
    showStartScreen()

    start_ticks=pygame.time.get_ticks()
    os.system('omxplayer  --win "0,0, 1600,500"  videotracteur.mp4&')
```

```
    while True:
        runGame()
        showGameOverScreen()
        seconds=(pygame.time.get_ticks()-start_ticks)/1000
        if (seconds > 75):
            terminate()
            return



def runGame():
    # Set a random start point.
    sleep(15)
    startx = random.randint(5, CELLWIDTH - 6)
    starty = random.randint(5, CELLHEIGHT - 6)
    wormCoords = [{'x': startx,     'y': starty},
                  {'x': startx - 1, 'y': starty},
                  {'x': startx - 2, 'y': starty}]
    direction = RIGHT

    # Start the apple in a random place.
    apple = getRandomLocation()

    while True: # main game loop
        sleep(0.15)
        buttonState17 = GPIO.input(17)
        buttonState18= GPIO.input(18)
        buttonState23 = GPIO.input(23)
        buttonState27= GPIO.input(27)
        buttonState22= GPIO.input(22)

        if (buttonState17 == False) and direction != UP:

            direction= DOWN
            print("Button Pressed 17")
        if (buttonState23 == False) and direction != DOWN:   #addd

            direction= UP
            print("Button Pressed 23")

        if (buttonState18 == False) and direction != RIGHT:

            direction= LEFT
            print("Button Pressed 18")
```

```
    if (buttonState27 == False) and direction != LEFT:  #add

        direction= RIGHT
        print("Button Pressed 27")

    buttonState22= GPIO.input(22)
    if (buttonState22 == False):          #add
        print("Button Pressed 22")
        return


    for event in pygame.event.get(): # event handling loop

        if event.type == QUIT:
            terminate()
        elif event.type == KEYDOWN:
            if (event.key == K_LEFT or event.key == K_a ) and direction != RIGHT:
                direction = LEFT
            elif (event.key == K_RIGHT or event.key == K_d) and direction != LEFT:
                direction = RIGHT
            elif (event.key == K_UP or event.key == K_w ) and direction != DOWN:
                direction = UP
            elif (event.key == K_DOWN or event.key == K_s) and direction != UP:
                direction = DOWN
            elif event.key == K_ESCAPE:
                terminate()

    # check if the worm has hit itself or the edge
    if wormCoords[HEAD]['x'] == -1 or wormCoords[HEAD]['x'] == CELLWIDTH or
wormCoords[HEAD]['y'] == -1 or wormCoords[HEAD]['y'] == CELLHEIGHT:
        return # game over
    for wormBody in wormCoords[1:]:
        if wormBody['x'] == wormCoords[HEAD]['x'] and wormBody['y'] ==
wormCoords[HEAD]['y']:
            return # game over

    # check if worm has eaten an apply
    if wormCoords[HEAD]['x'] == apple['x'] and wormCoords[HEAD]['y'] == apple['y']:
        # don't remove worm's tail segment
        apple = getRandomLocation() # set a new apple somewhere
    else:
        del wormCoords[-1] # remove worm's tail segment
```

```
        # move the worm by adding a segment in the direction it is moving

        if direction == UP:
            newHead = {'x': wormCoords[HEAD]['x'], 'y': wormCoords[HEAD]['y'] - 1}
        elif direction == DOWN:
            newHead = {'x': wormCoords[HEAD]['x'], 'y': wormCoords[HEAD]['y'] + 1}
        elif direction == LEFT:
            newHead = {'x': wormCoords[HEAD]['x'] - 1, 'y': wormCoords[HEAD]['y']}
        elif direction == RIGHT:
            newHead = {'x': wormCoords[HEAD]['x'] + 1, 'y': wormCoords[HEAD]['y']}
        wormCoords.insert(0, newHead)
        DISPLAYSURF.fill(BLACK)
        drawGrid()
        drawWorm(wormCoords)
        drawApple(apple)
        drawScore(len(wormCoords) - 3)
        pygame.display.update()
        FPSCLOCK.tick(FPS)

def drawPressKeyMsg():
    pressKeySurf = BASICFONT.render('Commencez le parcours!', True, BEIGE)
    pressKeyRect = pressKeySurf.get_rect()
    pressKeyRect.topleft = (WINDOWWIDTH - 200, WINDOWHEIGHT - 30)
    DISPLAYSURF.blit(pressKeySurf, pressKeyRect)




def checkForKeyPress():
    for event in pygame.event.get():
        if event.type == QUIT:      #event is quit
            terminate()
        elif event.type == KEYDOWN:
            if event.key == K_ESCAPE:   #event is escape key
                terminate()
            else:
                return event.key   #key found return with it
    # no quit or key events in queue so return None
    return None


def showStartScreen():
    titleFont = pygame.font.Font('freesansbold.ttf', 100)
    titleSurf1 = titleFont.render('Tracteur!', True, WHITE, DARKGREEN)
```

```
titleSurf2 = titleFont.render('Tractor!', True, GREEN)

degrees1 = 0
degrees2 = 0


pygame.event.get()  #clear out event queue
timestart=pygame.time.get_ticks()
while True:
   DISPLAYSURF.fill(BGCOLOR)
   rotatedSurf1 = pygame.transform.rotate(titleSurf1, degrees1)
   rotatedRect1 = rotatedSurf1.get_rect()
   rotatedRect1.center = (WINDOWWIDTH / 2, WINDOWHEIGHT / 2)
   DISPLAYSURF.blit(rotatedSurf1, rotatedRect1)

   rotatedSurf2 = pygame.transform.rotate(titleSurf2, degrees2)
   rotatedRect2 = rotatedSurf2.get_rect()
   rotatedRect2.center = (WINDOWWIDTH / 2, WINDOWHEIGHT / 2)
   DISPLAYSURF.blit(rotatedSurf2, rotatedRect2)

   drawPressKeyMsg()

   buttonState17 = GPIO.input(17)
   buttonState18= GPIO.input(18)
   buttonState23 = GPIO.input(23)
   buttonState27= GPIO.input(27)
   buttonState22= GPIO.input(22)

   if (buttonState17 == False) :
     return()
   if (buttonState23 == False) : return()

   if (buttonState18 == False) :return()

   if (buttonState27 == False) : return()

   buttonState22= GPIO.input(22)
   if (buttonState22 == False):          #add

      return()
   timenow=pygame.time.get_ticks()


   pygame.display.update()
```

```
        FPSCLOCK.tick(FPS)
        degrees1 += 3 # rotate by 3 degrees each frame
        degrees2 += 7 # rotate by 7 degrees each frame


def terminate():
   pygame.quit()
   sys.exit()


def restart():



    partieun()

def getRandomLocation():
    return {'x': random.randint(0, CELLWIDTH - 1), 'y': random.randint(0, CELLHEIGHT -
1)}


def showGameOverScreen():
    gameOverFont = pygame.font.Font('freesansbold.ttf', 150)
    gameSurf = gameOverFont.render('TRY AGAIN', True, WHITE)
    overSurf = gameOverFont.render('ESSAYE ENCORE', True, WHITE)
    gameRect = gameSurf.get_rect()
    overRect = overSurf.get_rect()
    gameRect.midtop = (WINDOWWIDTH / 2, 10)
    overRect.midtop = (WINDOWWIDTH / 2, gameRect.height + 10 + 25)

    DISPLAYSURF.blit(gameSurf, gameRect)
    DISPLAYSURF.blit(overSurf, overRect)
    drawPressKeyMsg()
    pygame.display.update()
  #  pygame.time.wait(1)#50

    pygame.event.get()  #clear out event queue
    while True:

        #if checkForKeyPress():
        pygame.time.wait(1)
        pygame.display.update()
       # os.system('omxplayer -o local sm64_mario_press_start.mp3')
        buttonState22= GPIO.input(22)
```

```
        if (buttonState22 == False):            #add
            print("Button Pressed 22")
            return


def drawScore(score):
    scoreSurf = BASICFONT.render('Score: %s' % (score), True, WHITE)
    scoreRect = scoreSurf.get_rect()
    scoreRect.topleft = (WINDOWWIDTH - 120, 10)
    DISPLAYSURF.blit(scoreSurf, scoreRect)


def drawWorm(wormCoords):
    for coord in wormCoords:
        x = coord['x'] * CELLSIZE
        y = coord['y'] * CELLSIZE
        wormSegmentRect = pygame.Rect(x, y, CELLSIZE, CELLSIZE)
        pygame.draw.rect(DISPLAYSURF, DARKGRAY, wormSegmentRect)
        wormInnerSegmentRect = pygame.Rect(x + 4, y + 4, CELLSIZE - 8, CELLSIZE - 8)
        pygame.draw.rect(DISPLAYSURF, YELLOW, wormInnerSegmentRect)


def drawApple(coord):


    x = coord['x'] * CELLSIZE
    y = coord['y'] * CELLSIZE
    appleRect =  pygame.Rect(x, y, CELLSIZE, CELLSIZE)
    pygame.draw.rect(DISPLAYSURF, YELLOW, appleRect)


def drawGrid():
    for x in range(0, WINDOWWIDTH, CELLSIZE): # draw vertical lines
        pygame.draw.line(DISPLAYSURF, GREEN, (x, 0), (x, WINDOWHEIGHT))
    for y in range(0, WINDOWHEIGHT, CELLSIZE): # draw horizontal lines
        pygame.draw.line(DISPLAYSURF, GREEN, (0, y), (WINDOWWIDTH, y))


if __name__ == '__main__':
    main()
```