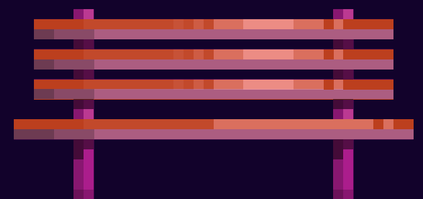
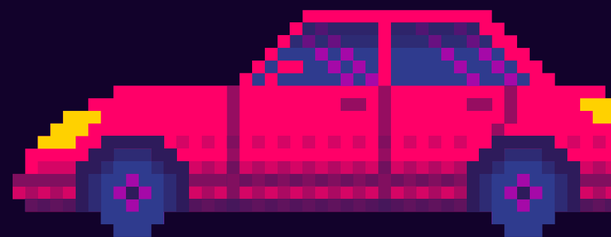
CATCH THE CAT

ASMAA BAZIGHE

WELCOME TO OUR PRESENTATION

1. CONCEPT OF THE GAME
2. TOOLS
3. THE CODE
4. THE GAME





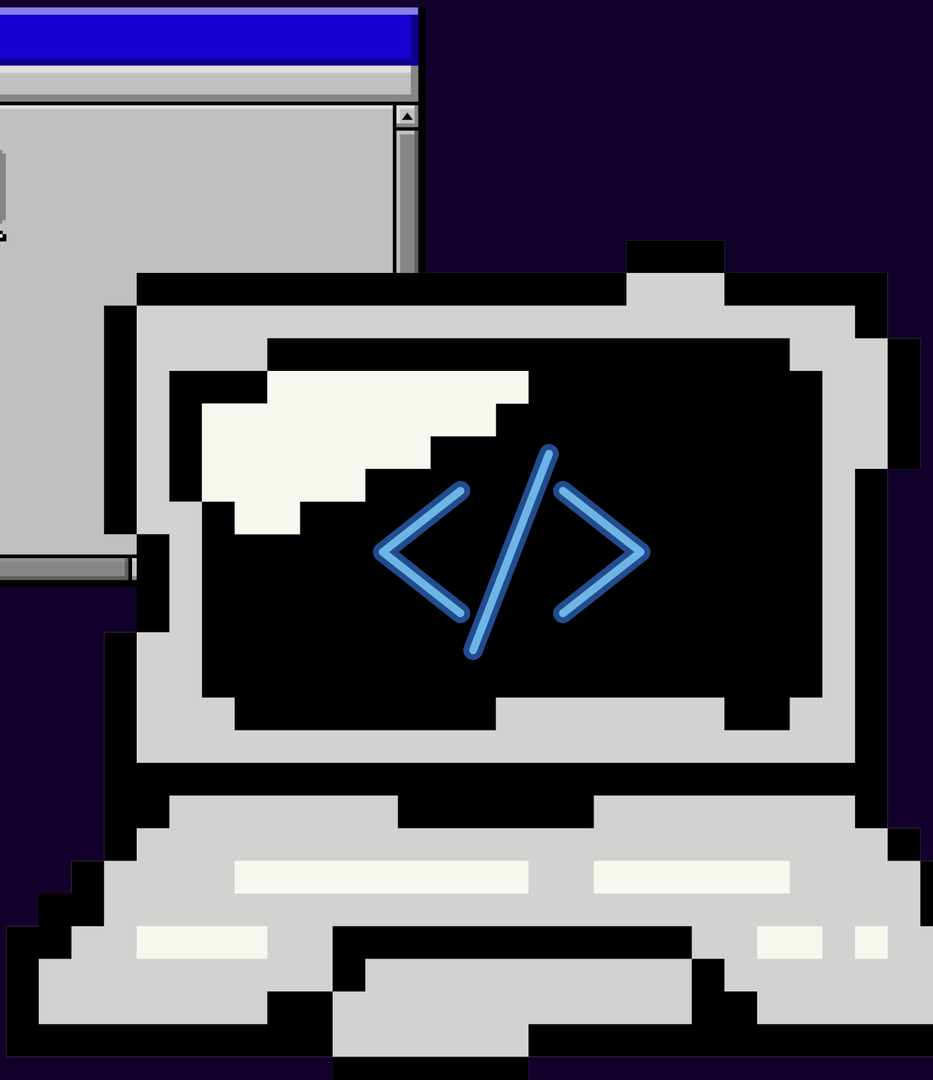
ABOUT GAME

Catch The Cat is an interactive board game that puts players' strategic prowess to the test as they endeavor to capture a cunning black cat attempting to flee the board. Set against a grid of white squares, players must strategically place barriers to block the cat's escape route, all while anticipating its unpredictable movements. With each turn presenting critical decision points, players must demonstrate strategic foresight and adaptability to outmaneuver their feline adversary and secure victory. Catch The Cat promises an exhilarating gaming experience, where every move counts and strategic mastery reigns supreme.

THE TOOLS

python High-level language facilitating rapid development. Large community and resources available.

Tkinter: Standard GUI toolkit for user interfaces in Python. Seamless integration with Python for creating interactive interfaces.



THE CODE

IMPORTING TKINTER:

```
1       import tkinter as tk #i
```

This line imports the Tkinter library, which is used for creating graphical user interfaces.

INITIALIZING VARIABLES

```
# Initialiser une liste pour garder trace des cases cochées
cases_cochees = []
```

This initializes an empty list cases_cochees to keep track of checked squares.

DEFINING DESSINER_PLATEAU FUNCTION:

```python
def dessiner_plateau():
    #Déclaration des variables globales
    global cercle_id  # Identifiant du cercle noir au centre du plateau
    global cases_blanches  # Liste pour stocker les coordonnées des cases blanches
    global cases_vertes  # Liste pour stocker les coordonnées des cases vertes

    cases_blanches = []  # Liste pour stocker les coordonnées des cases blanches
    cases_vertes = [] # Liste pour stocker les coordonnées des cases vertes


    for i in range(11):# Boucle pour parcourir les lignes du plateau
        for j in range(15):# Boucle pour parcourir les colonnes du plateau
            if i == 0 or i == 10 or j == 0 or j == 14: # Si la case est sur le bord du plateau
                #Rectangle:
                #canvas.create_rectangle(x1, y1, x2, y2,.....)
                #(x1,y1) : top left     (x2,y2): bottom right
                canvas.create_rectangle(j*50, i*50, (j+1)*50, (i+1)*50, fill='green', outline='green')
                cases_vertes.append((i, j)) # Ajouter les coordonnées de la case verte à la liste
            else:
                # Dessiner une case blanche pour les cases internes du plateau
                rect_id = canvas.create_rectangle(j*50, i*50, (j+1)*50, (i+1)*50, fill='white', outline='black')
                cases_blanches.append((i, j))  # Ajouter les coordonnées de la case blanche

                # Si la case est à la position spécifique (5, 7), dessiner un cercle noir au centre
                if i == 5 and j == 7:
                    cercle_id = canvas.create_oval(j*50+10, i*50+10, (j+1)*50-10, (i+1)*50-10, fill='black')
```

DEFINING DESSINER_PLATEAU FUNCTION:

This function is responsible for drawing the game board.

- It initializes global variables cercle_id, cases_blanches, and cases_vertes.
- It iterates over the rows and columns of the board and draws rectangles using the Canvas widget from Tkinter. Green rectangles represent the border, while white rectangles represent the inner squares. The coordinates of green and white squares are stored in cases_vertes and cases_blanches, respectively.

DEFINING COCHER_CASE FUNCTION:

```
def cocher_case(event):
    global cercle_id
    x, y = event.x, event.y # Récupérer les coordonnées x et y du clic de
    i, j = y // 50, x // 50 # Convertir les coordonnées en indices de ligr
    if 0 < i < 10 and 0 < j < 14 and (i,j)!=(ni,nj): # Vérifier si les inc
        if (i, j) not in cases_cochees :  # Vérifier si la case n'est pas
            if (i, j) in cases_blanches :  # Vérifier si la case est blanc
                cases_cochees.append((i, j)) # Ajouter les coordonnées de
                canvas.itemconfig(canvas.find_closest(x, y), fill='red')
                deplacer_cercle_minimax() # Appeler la fonction pour dépla
```

DEFINING COCHER_CASE FUNCTION:

This function handles mouse clicks on the canvas.

- It retrieves the coordinates of the click and converts them into row and column indices.
- If the clicked square is within the valid range of the board and not already checked, it adds the coordinates to cases_cochees, changes the color of the square to red, and calls the deplacer_cercle_minimax function.

GLOBAL VARIABLES INITIALIZATION:

```
global i
global j

i, j = 5, 7 #coordonné initial du cercle
```

These lines initialize global variables i and j to the starting coordinates of the black circle.

DEFINING EVALUER_ETAT FUNCTION:

```python
def evaluer_etat(etat):
    # Coordonnées du centre fixe
    centre_x = 350
    centre_y = 250
    # Coordonnées du cercle (état)
    cercle_x, cercle_y = etat
    # Calcul de la distance euclidienne entre le centre et le cercle
    distance = ((centre_x - cercle_x) ** 2 + (centre_y - cercle_y) ** 2) ** 0.5
    # Retourne la valeur négative de la distance
    return -distance
```

This function calculates the Euclidean distance between the fixed center and the circle's coordinates and returns the negative distance.

DEFINING ACTIONS_POSSIBLES FUNCTION:

```
def actions_possibles(etat):
    # Déplacements possibles en x et y
    dx = [-1, 0, 1, 0, -1, 1, -1, 1]
    dy = [0, -1, 0, 1, -1, -1, 1, 1]
    x, y = etat# Coordonnées de l'état
    actions = [(x+dx[i], y+dy[i]) for i in range(8)] # Générer les nc

    # Filtrer les actions pour ne conserver que celles qui sont dans
    return [(i, j) for i, j in actions if (i, j) in cases_blanches]
```

This function generates possible moves for the circle based on the current state, filtering out moves that go outside the white squares.

DEFINING MINIMAX FUNCTION:

```
def minimax(etat, profondeur, alpha, beta, maximizing):
    if profondeur == 0: # Cas de base : profondeur atteinte, retourner l'évaluation de l'état
        return evaluer_etat(etat)

    # Cas où l'on maximise la valeur
    if maximizing:
        valeur = float('-inf') # Initialisation de la valeur par moins l'infini


        # Parcours de toutes les actions possibles à partir de l'état actuel
        for action in actions_possibles(etat):
            # Appel récursif pour évaluer l'action et mettre à jour la valeur maximale
            valeur = max(valeur, minimax(action, profondeur-1, alpha, beta, False))
            # Mise à jour de la valeur alpha pour l'élagage alpha-bêta
            alpha = max(alpha, valeur)
            # algo alpha-bêta : arrêt de la recherche si beta <= alpha
            if beta <= alpha:
                break

        # Retourne la valeur maximale trouvée
        return valeur
    else:
        # Cas où l'on minimise la valeur
        valeur = float('inf') # Initialisation de la valeur par l'infini

    for action in actions_possibles(etat):
        # Appel récursif pour évaluer l'action et mettre à jour la valeur minim
        valeur = min(valeur, minimax(action, profondeur-1, alpha, beta, True))

        # Mise à jour de la valeur beta pour l'élagage alpha-bêta
        beta = min(beta, valeur)

        # Élagage alpha-bêta : arrêt de la recherche si beta <= alpha
        if beta <= alpha:
            break
    # Retourne la valeur minimale trouvée
    return valeur
```

DEFINING MINIMAX FUNCTION:

This function implements the minimax algorithm to evaluate possible future states of the game and select the best move for the circle.

DEFINING DEPLACER_CERCLE_MINIMAX FUNCTION:

```
def deplacer_cercle_minimax():
    # Déclarations des variables globales utilisées dans la fonction
    global cercle_id
    global cases_blanches
    global i
    global j
    global cases_cochees
    global ni
    global nj
    # Directions possibles pour le déplacement du cercle
    directions = [(0, 1), (0, -1), (1, 0), (-1, 0), (1, 1), (1, -1), (-1, 1), (-1, -1)]

    # Initialisation des meilleures actions et de la meilleure valeur
    meilleures_actions = []
    meilleure_valeur = float('-inf')

    # Boucle à travers les directions possibles
    for dx, dy in directions:
        ni, nj = i + dx, j + dy
        if (ni, nj) not in cases_cochees: # Vérifier si la case n'est pas déjà cochée
            # Évaluation de l'état résultant par l'algorithme minimax
            valeur = minimax((ni, nj), profondeur=3, alpha=float('-inf'), beta=float('inf'), maximizing=False)

            # Mettre à jour les meilleures actions si une valeur meilleure est trouvée
            if valeur > meilleure_valeur:
             meilleure_valeur = valeur
             meilleures_actions = [(ni, nj)]
         elif valeur == meilleure_valeur:
             meilleures_actions.append((ni, nj))

# Si des meilleures actions sont trouvées
if meilleures_actions:
    ni, nj = meilleures_actions[0]
    # Supprimer le cercle actuel et créer un nouveau cercle à la meilleure position
    canvas.delete(cercle_id)
    cercle_id = canvas.create_oval(nj*50+10, ni*50+10, (nj+1)*50-10, (ni+1)*50-10, fill='black')

    # Vérifier si le nouveau cercle est sur une case verte
    if (ni, nj) in cases_vertes:
        canvas.create_text(375, 525, text="Le chat a échappé !", fill="black", font=("Helvetica", 12))
    else:
        # Mettre à jour les coordonnées i, j
        i, j = ni, nj
else:
    # Si aucune meilleure action n'est trouvée
    canvas.create_text(375, 525, text="vous avez réussi à attraper le chat", fill="black", font=("Helvetica", 12))
```

DEFINING DEPLACER_CERCLE_MINIMAX FUNCTION:

This function moves the circle according to the minimax algorithm's decision.

- It calculates the best move using the minimax algorithm, updates the circle's position, and redraws it on the canvas.
- If the circle reaches a green square, it announces the cat's victory.

CREATING THE GUI

```
# Créer une fenêtre
fenetre = tk.Tk()
fenetre.title("Plateau de jeu")

# Créer un canevas pour dessiner le plateau
canvas = tk.Canvas(fenetre, width=750, height=550)
canvas.pack()

# Dessiner le plateau
dessiner_plateau()

# Lier la fonction cocher_case à un clic de souris
canvas.bind("<Button-1>", cocher_case)

# Démarrer la boucle principale
fenetre.mainloop()
```

- Initializes a Tkinter window and a canvas for drawing the game board.
- Draws the game board using the dessiner_plateau function.
- Binds mouse clicks to the cocher_case function.
- Starts the main event loop to handle user interactions (fenetre.mainloop()).

This code implements a game where the player clicks on white squares to trap a black circle (representing a cat) within the board, using strategic placement of barriers. The game logic involves evaluating possible moves using the minimax algorithm to ensure an engaging and challenging gameplay experience.

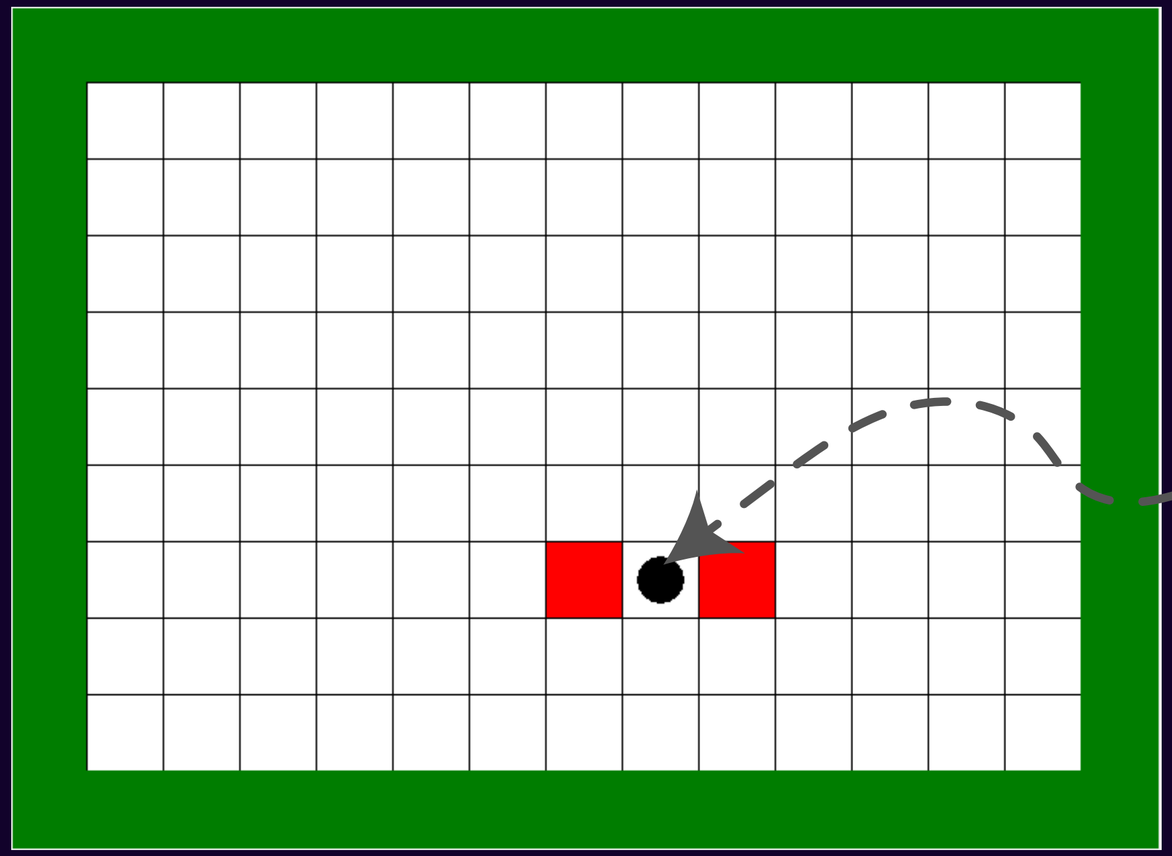
THE GAME

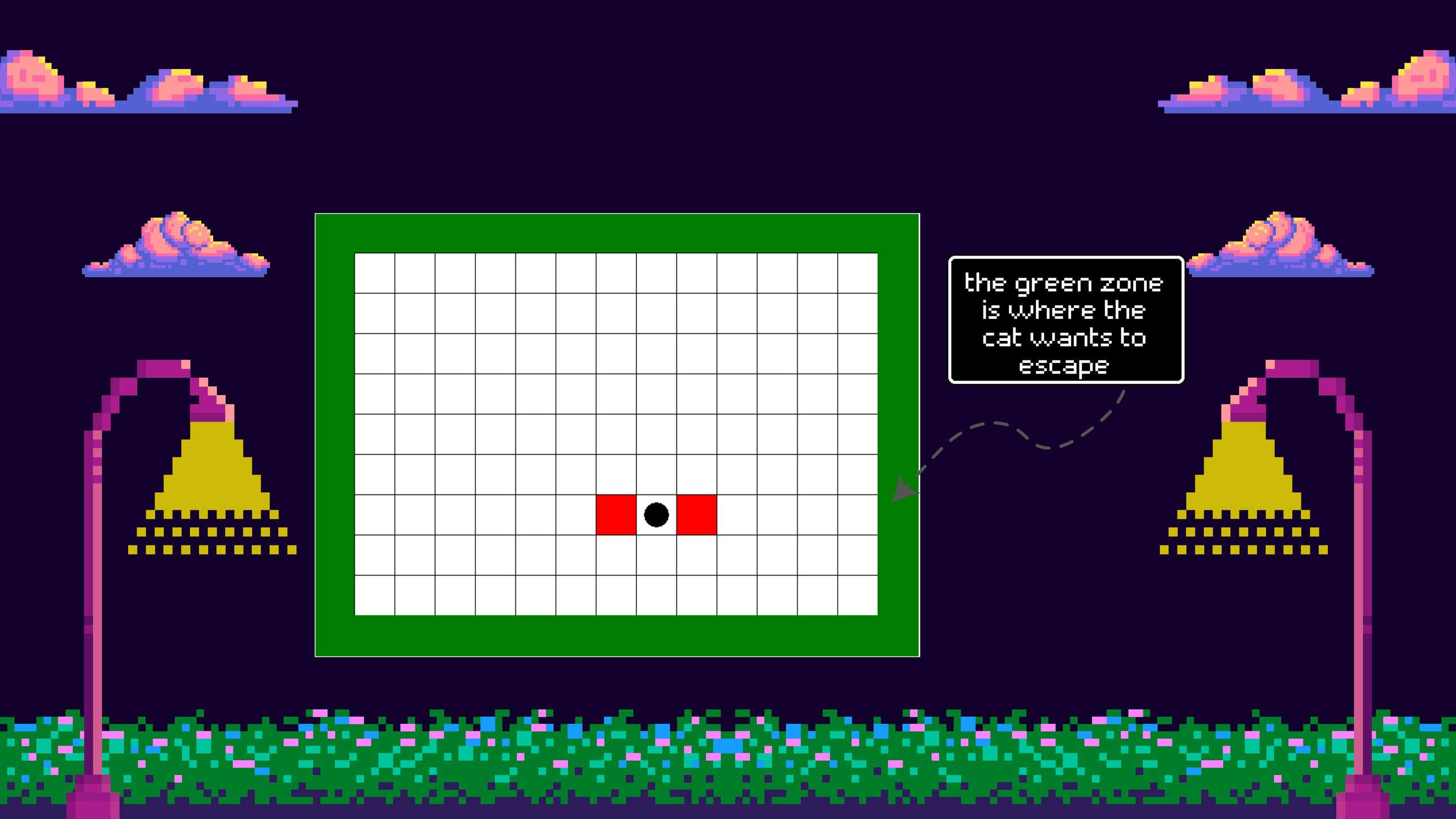
01 Capture the Cat: The primary objective of the player is to capture the black cat trying to escape the board.

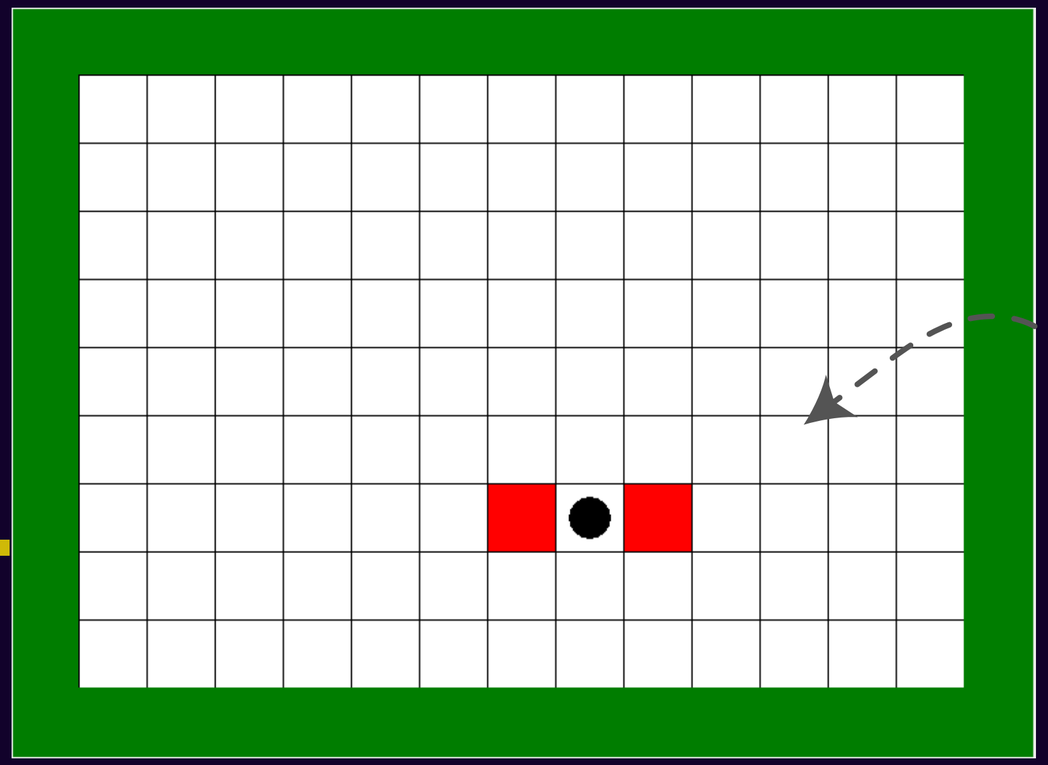
02 Blocking Movement: Players can strategically block the cat's movement by creating barriers on the board.

03 Anticipate Movements: Players must anticipate the cat's movements to effectively trap it within the board.

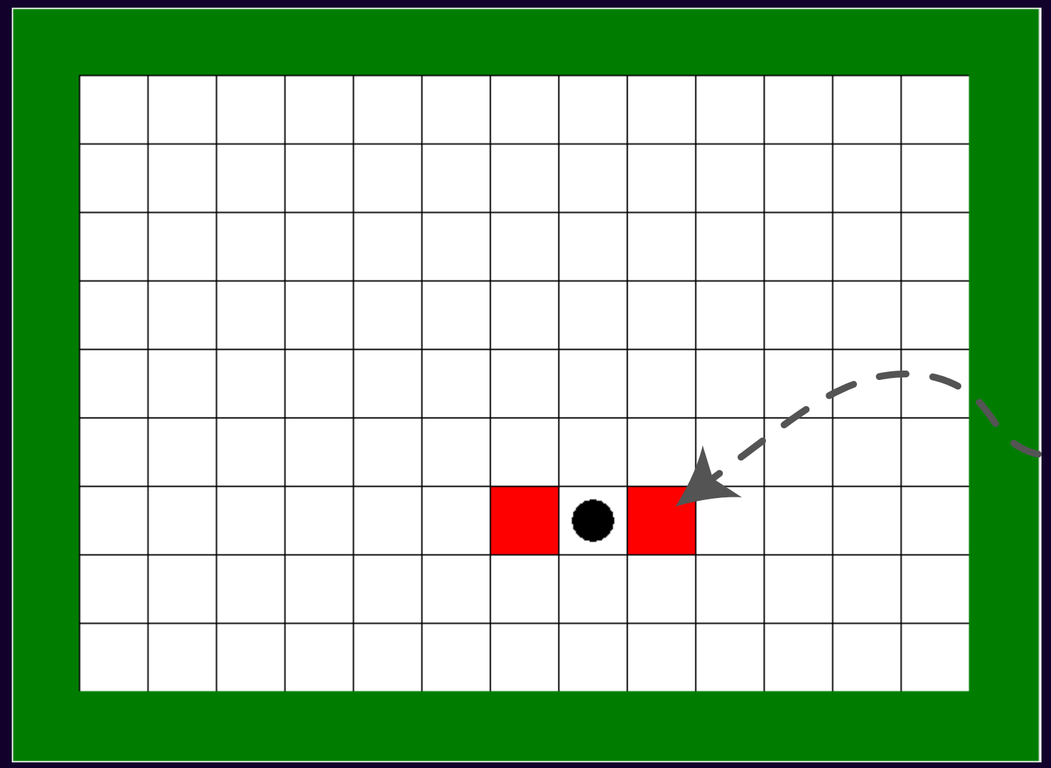
04 Avoid Green Squares: The cat's victory condition is to reach a green square, so players must prevent it from doing so to win.


the cat is represented by a black circle

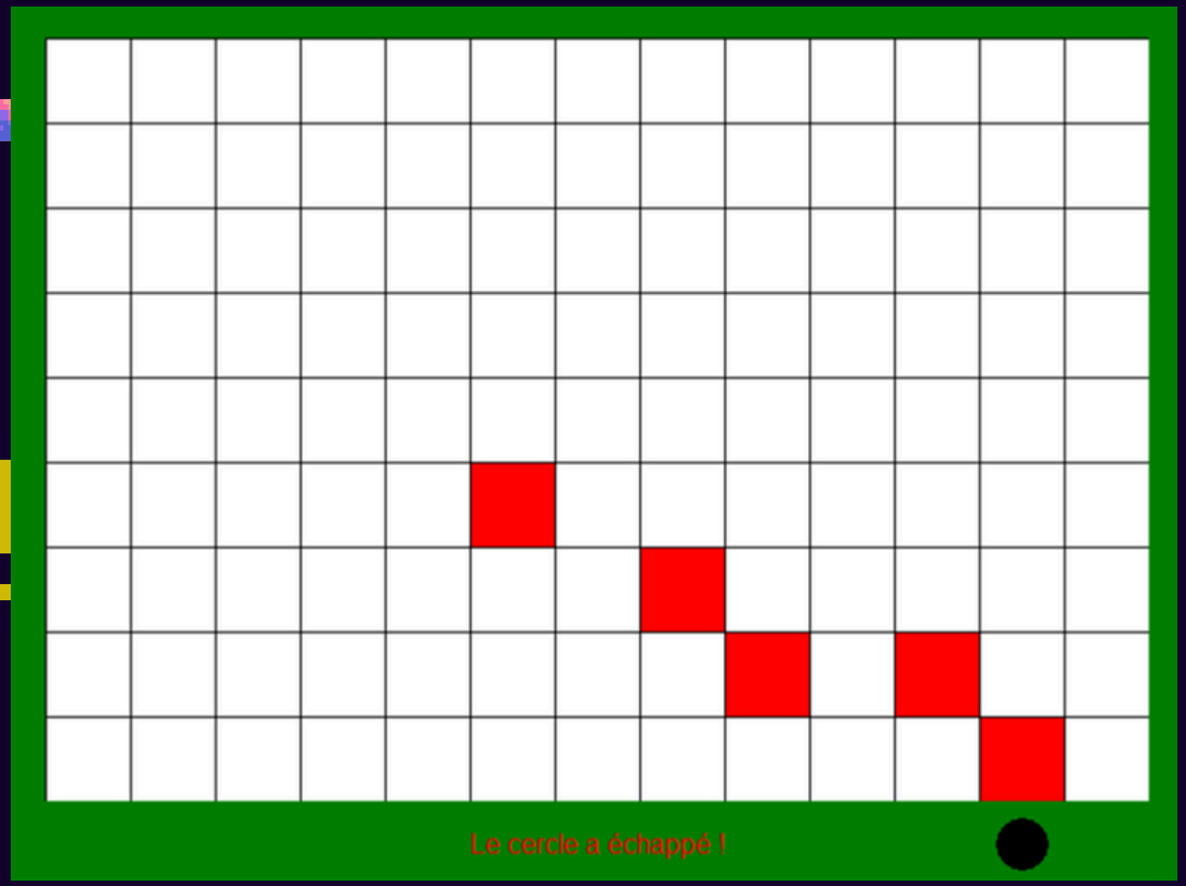

the green zone is where the cat wants to escape



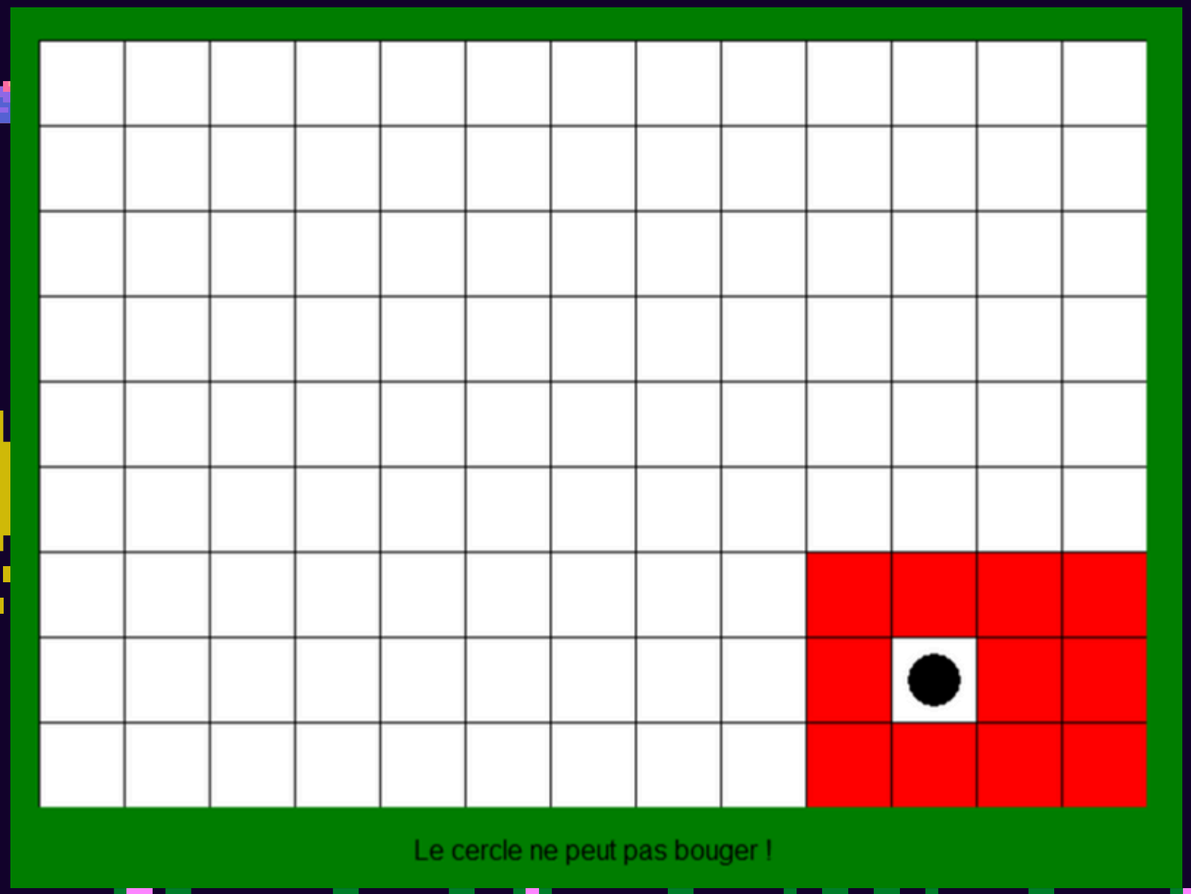
the white zone is the area of the game, i.e. where the player will click to prevent the cat from running away



The color changes to red to indicate checked boxes. they represent an obstacle for the cat.



If the player fails to correctly anticipate the black cat's movements, it will manage to escape, and the player will lose the game.



Le cercle ne peut pas bouger !

On the other hand, if the player manages to successfully predict the movements of the black cat and manages to capture it, he will win the game.

HERE'S WHERE WE GOT TO WHEN IT COMES TO GRAPHICAL ENHANCEMENT.


THANK
YOU