* Our project consists of two engaging and educational games designed to enhance children's cognitive skills. The first game focuses on teaching numbers through a car-themed adventure, while the second game aims to educate children about shapes and colors through an interactive puzzle challenge.

**MENU**

* The menu allows movement between the games or exiting the game by clicking on buttons with the mouse.

**CAR GAME**

* We designed a car game whose purpose is teaching the children counting from one to nine with a car.
* The game consists of 3 obstacles, which are numbers on the road, one car, one scoreboard, and one table which demonstrates the images on which counting is based.
* If the value of the obstacle and the shown number on the table during the reaction between the car and one of the obstacles players acquire 50 points. If not, the car game will be closed, and the scene turn into the game over scene of the car game.
* Also, players gain 0.1 points per tick as long as the car game is executed.
* Correct hits prompt the drone to reevaluate and reveal the targeted number.

Gameplay Overview:

* Count fun groups of shapes in the card game.
* Swiftly hit the correct number with the car.
* Quick reactions are crucial due to varying shapes and speeds.
* Incorrect hits lead to game over.
* Correct hits prompt the drone to reevaluate and reveal the targeted number.

**GAME OVER SCENE OF CAR GAME**

* It demonstrates the last score that is taken from the car game until players lose the game.
* Moreover, there are two buttons on the game over scene of the car game.
* The left button allows that player goes the menu.
* The right button allows that player play the car game again.

**PUZZLE GAME**

* We design a puzzle game. We cut 4 shapes in 4 different colours for one of our pictures. We wrote that shape and colour in each picture space we cut.
* The game consists of 4 objects and 1 background. You win when you bring the objects to the right place and ensure the integrity of the picture. You use the mouse to move these objects.

1-) Square - 220 - Green

2-) Rectangle - 300 100 - Yellow

3-) Circle – 110 - Red

4-) Triangle - 220 - Blue

**GAME OVER SCENE OF PUZZLE GAME**

* There are two buttons on the game over scene of the puzzle game.
* The left button allows that player goes the menu.
* The right button allows that player play the puzzle game again.

 Gameplay Overview:

* Learn shapes and colors in the puzzle game.
* Match each shape with its assigned color.
* Complete the picture by placing puzzle pieces correctly.
* Validate the arrangement by pressing a tick mark.
* Successful completion leads to victory; incorrect arrangement prolongs the game.
* Our educational game project combines entertainment with learning, providing children with an interactive platform to grasp numerical concepts through the car game and understand shapes and colors in the puzzle game. Both games are designed to stimulate cognitive development in a fun and engaging way, fostering an enjoyable learning experience for young participants.

=============BAYRAM'S PART=============

* Coding puzzle game, designing, and editing the visuals of the game such as scenes, scoreboard images, and so on.

MOUSE + GAME-SCENE + DRAGGİNG:

; purpose: check whether the point on the button or not

;contract: isInButton: b(button) x(number) y(number) -> boolean

isInButton🡪 control mouse tip point

; purpose: check whether mouse clicked button or not

; contract: doesClickButton: b(button) x(number) y(number) mo(mouseEvent)-> boolean

doesClickButton🡪To check whether the mouse clicked and grabbed the object

; purpose: check whether the point on the rectangle or not

; contract: isInRectangle: rec(rectangleOb) x(number) y(number) -> boolean

isInRectangle🡪 To check whether the mouse tip is on its rectangle or not

; purpose: check whether the point on the circle or not

; contract: isInCircle: circ(circleOb) x(number) y(number) -> boolean

isInCircle🡪 To check whether the mouse tip is on its circle or not

; purpose: check whether the point on the triangle or not

; contract: isInTriangle: tri(triangleOb) x(number) y(number) -> boolean

isInTriangle🡪 To check whether the mouse tip is on its triangle or not

; purpose: check whether the point on the square or not

; contract: isInTriangle: sq(squareOb) x(number) y(number) -> boolean

isInSquare🡪 To check whether the mouse tip is on its square or not

; purpose: check whether mouse is dragged or not

; contract: doesDrag: pg(puzzleGame) x(number) y(number) mo(mouseEvent)-> boolean

doesDrag🡪 We wrote a mouse move function to hold and drag objects with the mouse and drop them into the correct space.

; purpose: changing between the scenes by clicking mouse1 and dragging the shapes in the puzzle game

; contract: mouseUpdate: ga(GAME) x(number) y(number) mo(mouseEvent)-> GAME

mouseUpdate🡪 I did half of this function. Generally, I code dragging parts

PUZZLE REACTİONS:

; purpose: calculating the distance between the positions

; contract: distCalc: pos1(position) pos2(position) -> number

distCalc🡪 We wrote this code to measure the distance between the initial location and the target location.

; purpose: determinating whether puzzle are completed or not

; contract: doesFillUp: sq(squareOb) triang(trinagleOb) rect(rectangleOb) circ(circleOb) -> boolean

doesFillUp🡪 To check whether the puzzle has all the objects in the correct position. To understand whether a picture has been created

PUZZLE GAME BACKGROUND DESIGNING:

; purpose: drawing the puzzle game

; contract: finalScenePuzzle: ga(puzzleGame) -> image

finalScenePuzzle🡪to create the puzzle game with a background and draw the game

==============YUNUS'S PART==============

* Coding half of the car game and designing background of the car game such as sky, road and roadmarks.

CAR OBJECT:

; purpose: update the position of the car

; contract: newPosCar: c(theCar) key(key) 🡪 position

; purpose: main function for moving the car object

; contract: moveCar c(theCar) k(key) 🡪 car

moveCar🡪 position change which base on keyboard buttons

OBSTACLE OBJECT:

; purpose: update the position with the velocity

; contract: newPosObs: pos(position) vel(velocity) 🡪 position

newPosObs🡪 movement of the obstacle with the velocity

; purpose: designing obstacle images

; contract: designImObs: obs(obstacle)-> image

designImObs🡪 creating the images of the obstacles

; purpose: determinating whether the obstacle in frame or not

; contract: inFrameX: obs(obstacle) -> boolean

inFrameX🡪 checking obstacles is out of x-axis or not

FIGURE OBJECT:

; purpose: updating figure

; contract: updateFigure: c(theCar) fig(figure) obs1(obstacle) obs2(obstacle) obs3(obstacle) -> figure

updateFigure🡪 updating figure after the correct answers from the player

; purpose: creating image of the figures with correct answers

; contract: textFig: c(theCar) fig(figure) obs1(obstacle) obs2(obstacle) obs3(obstacle) -> image

textFig🡪 after updating figure creating image of the obstacle

; purpose: showing the figure

; contract: figureImage: gc(gameCar) -> image

figureImage🡪 placing the figure on the table

REACTİON BETWEEN CAR AND OBSTACLE:

; purpose: check whether car and obstacle are collided or not

; contract: isCollided: c(car) ob(obstacle) -> boolean

isCollided🡪 If the coordinates of the car and the obstacle match, the function returns #true; otherwise, it returns #false.

CAR GAME SCORE AND SCOREBOARD:

; purpose: update the game score with interaction between car and right obstacle

; contract: updateScore: g(gameCar) -> number

; purpose: showing the score

; contract: board: g(gameCar) -> image

updateScore-board🡪 creating the score and scoreboard on the car game

==============UMUT'S PART==============

OBSTACLE OBJECTS:

; purpose: updating velocities, positions, value of obstacle, and number of figures with each new tour

; contract: updateVelObs: c(theCar) obs(obstacle) fig(figure) -> obstacle

updateVelObs🡪 If the choice of the car is correct or the obstacle is out of the frame the obstacle goes to the start line and its velocity changes randomly. Furthermore, if the choice of the car is correct figure changes randomly as well.

CAR GAME GAME-OVER SCENE:

; purpose: showing the score on car game-over scene

; contract: OverSceneDraw: gOvSceCar(gameOverSceneCar) gaC(gameCar) -> image

OverSceneDraw🡪designing game-over scene of the car game and adding the acquired score on the game-over scene.

CAR GAME IMAGE UPDATES:

; purpose: drawing the car into the finalScene

; contract: scene1: g(gameCar) -> image

; purpose: adding the obtacles on scene1

; contract: scene2: g(gameCar) -> image

; purpose: adding the scoreboard and the figure on scene2

; contract: finalSceneCar: g(gameCar) -> image

scene1-scene2-finalSceneCar🡪 creating final scene of the car game

MOUSE + GAME-SCENE + DRAGGİNG:

; purpose: changing between the scenes by clicking mouse1 and dragging the shapes in the puzzle game

; contract: mouseUpdate: ga(GAME) x(number) y(number) mo(mouseEvent)-> GAME

mouseUpdate🡪 I did half of this function. Generally, I code clicking the button and changing the scene part.

; purpose: changing car game scene to game over scene of the car due to wrong answer

; contract: sceneChanger: ga(GAME) -> string

sceneChanger🡪After the clicking buttons with mouse1 scene of the game change

MAIN GAME UPDATE FUNCTIONS:

; purpose: updating the game by user's key input

; contract: updateKeyCar: g(GAME) k(key) -> GAME

updateKeyCar🡪 It provides only movements of the car from the car game with “w” “s” “up” and “down”.

; purpose: update the game with respect to time

; contract: updateTickCar: ga(GAME) -> GAME

updateTickCar🡪 It provides movement of the obstacles, change of figure and obstacles’ velocities and so on with each tick.

; purpose: draw the game into a scene

; contract: finalDraw: game(GAME) -> image

finalDraw🡪 final images, scenes of the game