\*\*\* major changes in WinningConditions script… best to apply your own changes on this script rather than to add these changes to your current script \*\*\*

using UnityEngine;

using System.Collections;

using System.Collections.Generic;

/// <summary> ##################################

///

/// NOTICE :

/// This script is conditions set to win/end the current game.

///

/// </summary> ##################################

public class JSFWinningConditions : MonoBehaviour {

[Tooltip("Starts the game running when scene loads.\n" +

"call 'startThisGame()' function yourself if you have other plans.")]

public bool startGameImmediately = true;

public float checkSpeed = 1;

public bool specialTheLeftovers = true;

public float secondsPerSpecial = 5;

public int movesPerSpecial = 5;

public bool popSpecialsBeforeEnd = true;

[Space(20)] // just some seperation

// timer game

public bool isTimerGame = false;

public TextMesh timeLabel;

public TextMesh timeText;

public float TimeGiven = 120;

[Space(20)] // just some seperation

// max move game

public bool isMaxMovesGame = false;

public TextMesh movesLabel;

public TextMesh movesText;

public int allowedMoves = 40;

[Space(20)] // just some seperation

// score game

public bool isScoreGame = true;

[Tooltip("If enabled, player must obtain a minimum score of 'scoreToReach'")]

public bool scoreRequiredWin = true;

[Tooltip("If enabled, obtaining the minimum score will trigger end-game.")]

public bool scoreEndsGame = false;

public int scoreToReach = 10000;

public int scoreMilestone2 = 20000;

public int scoreMilestone3 = 30000;

[Space(20)] // just some seperation

// clear shaded game

public bool isClearShadedGame = false;

[Tooltip("If enabled, player must clear all the shaded panels to win.")]

public bool shadesRequiredWin = true;

[Tooltip("If enabled, clearing all the shaded panels will trigger end-game.")]

public bool shadesEndsGame = true;

int shadesLeft = 1;

[Space(20)] // just some seperation

// get type game

public bool isGetTypesGame = false;

[Tooltip("If enabled, player must get all the specified types to win.")]

public bool typesRequiredWin = true;

[Tooltip("If enabled, getting all the specified types will trigger end-game.")]

public bool typeEndsGame = true;

public int[]numToGet = new int[9];

public GameObject placeholderPanel;

public GameObject textHolder;

TextMesh[] desc = new TextMesh[9];

bool collectedAll = false;

[Space(20)] // just some seperation

// treasure game

public bool isTreasureGame = false;

[Tooltip("If enabled, player must all the treasures to win.")]

public bool treasureRequiredWin = true;

[Tooltip("If enabled, getting all the treasures will trigger end-game.")]

public bool treasureEndsGame = true;

public TextMesh treasureLabel;

public TextMesh treasureText;

public int numOfTreasures = 3;

public int maxOnScreen = 2;

[Range(0,30)]public int chanceToSpawn = 10;

public List<Vector2> treasureGoal = new List<Vector2>();

public List<JSFGamePiece> treasureList = new List<JSFGamePiece>();

[HideInInspector] public int treasuresCollected = 0;

[HideInInspector] public int treasuresSpawned = 0;

public GameObject GameOverMessage;

JSFGameManager gm {get{return JSFUtils.gm;}}

float timeKeeper = 0; // just an in-game timer to find out how long the round has been playing..

bool isGameOver = false;

/// <summary>

///

/// Below are properties of interest...

///

/// gm.score <--- the current score accumulated by the player

/// gm.moves <--- the total moves the player has made

/// gm.currentCombo <--- the current combo count of any given move ( will reset to 0 each move )

/// gm.maxCombo <--- the max combo the player has achieved in the gaming round

/// gm.gameState <--- the status of the GameManager

/// gm.checkedPossibleMove <--- a boolean that signifies the board has stabilized from the last move

/// ( use this when you want the board to stop only after finish combo-ing )

/// gm.canMove <--- a boolean to allow players to move the pieces. true = can move; false = cannot move.

/// gm.board[x,y] <--- use this to reference the board if you needed more board properties

/// gm.notifyBoardHasChanged() <--- to tell the board to continue checks after it has settled

/// gm.matchCount[x] <--- the count of the type that has been destroyed.

///

/// </summary>

#region routines & related

IEnumerator updateStatus(){

while(gm.gameState != JSFGameState.GameOver) {// loop infinitely until game over

// updates the status...

if(isTimerGame){

if(timeText != null){

if ((TimeGiven - timeKeeper) >= 0){

timeText.text = (TimeGiven - timeKeeper).ToString(); // outputs the time to the text label

}else {

timeText.text = "0"; // outputs the time to the text label

}

}

}

// if(isScoreGame){

// // score is handled by GameManager and VisualManager

// }

if(isMaxMovesGame){

if(movesText != null){

if ((allowedMoves - gm.moves) >= 0){

movesText.text = (allowedMoves - gm.moves).ToString(); // outputs the time to the text label

}else {

movesText.text = "0"; // outputs the time to the text label

}

}

}

if(isClearShadedGame){ // updates the 'shadesLeft' variable...

shadesLeft = 0;

for ( int x = 0; x < gm.boardWidth; x++){

for (int y = 0; y < gm.boardHeight; y++){

if(gm.board[x,y].panel.pnd is JSFShadedPanel){

shadesLeft+= gm.board[x,y].panel.durability+1; // increase count as this is a shaded panel

}

}

}

}

if(isGetTypesGame){ // updates the 'collectedAll' variable...

collectedAll = true;

for(int x = 0; x < gm.pieceTypes.Length;x++){

if(numToGet[x] > 0 && x < gm.NumOfActiveType){

int val = numToGet[x] - gm.matchCount[x]; // num of remaining pieces to collect

if(val > 0){

desc[x].text = val.ToString() + " left";

}else{

desc[x].text = "0 left";

}

}

if(x < gm.NumOfActiveType && !(gm.matchCount[x] >= numToGet[x] ) ){

collectedAll = false; // still got pieces to collect...

}

}

}

// function to collect treasure as well as update the status...

if(isTreasureGame){

foreach(Vector2 pos in treasureGoal){

foreach(JSFGamePiece gp in treasureList){ // loop each treasure piece

Vector2 temp = new Vector2(gp.master.arrayRef[0],gp.master.arrayRef[1]);

if(temp == pos && !gp.master.isFalling){

treasuresCollected++; // increase collected count

gp.pd.onPieceDestroyed(gp); // the destroy call for treasure object

gp.removePiece(); // destroy the treasure

treasureList.Remove(gp); // remove from the list

break;

}

}

}

if(treasureText != null){

treasureText.text = (numOfTreasures - treasuresCollected).ToString();

}

}

yield return new WaitForSeconds(checkSpeed); // wait for the refresh speed

}

}

IEnumerator routineCheck(){

while(!isGameOver) {// loop infinitely until game over

// perform the checks

if(isTimerGame){

checkTime();

}

if(isMaxMovesGame){

checkMoves();

}

if(isScoreGame && scoreEndsGame){

checkScore();

}

if(isClearShadedGame && shadesEndsGame){

checkShaded();

}

if(isGetTypesGame && typeEndsGame){

checkNumsOfType();

}

if(isTreasureGame && treasureEndsGame){

checkTreasures();

}

yield return new WaitForSeconds(checkSpeed); // wait for the refresh speed

}

}

IEnumerator timer(){

while(!isGameOver) {// loop infinitely until game over

timeKeeper++; // timer increase in time

yield return new WaitForSeconds(1f); // ticks every second...

}

}

// function to check the time

void checkTime(){

if( TimeGiven <= timeKeeper ){

StartCoroutine(gameOver());

}

}

// function to compare score

void checkScore(){

if( gm.score > scoreToReach ){

StartCoroutine(gameOver());

}

}

// function to compare moves left

void checkMoves(){

if( gm.moves >= allowedMoves){

StartCoroutine(gameOver());

}

}

// function to check whether there are any shaded panels left...

void checkShaded(){

if( shadesLeft == 0){ // when no shaded panels are found, game over

StartCoroutine(gameOver());

}

}

// function to check whether the number of types to get is reached...

void checkNumsOfType(){

if(collectedAll){

StartCoroutine(gameOver()); // collected all, initiate game over

}

}

// function to check that the player has collected all treasures

void checkTreasures(){

if(treasuresCollected == numOfTreasures){

StartCoroutine(gameOver()); // collected all, initiate game over

}

}

#endregion routines & related

#region endgame sequence

IEnumerator gameOver(){

gm.audioScript.gameOverSoundFx.play(); // play the game over fx

gm.canMove = false; // player not allowed to move anymore

gm.gameState = JSFGameState.GameFinalizing; // game in finalizing mode...

isGameOver = true; // game over, all routine loops will be disabled

yield return new WaitForSeconds(1f); // wait for board to finish its routine actions

if(specialTheLeftovers){

while(gm.checkedPossibleMove == false){

// pause here till board has finished stabilizing...

yield return new WaitForSeconds(0.5f); // just to calm down from being so fast...

}

if(isTimerGame){

while( convertTime() ){ // converts time every second until no more time.

yield return new WaitForSeconds(0.5f);

}

}

if(isMaxMovesGame){

while(convertMoves() ){ // converts moves every second until no more moves.

yield return new WaitForSeconds(0.5f);

}

}

}

if(popSpecialsBeforeEnd){ // the feature is enabled

while(true){

while(gm.checkedPossibleMove == false){

// pause here till board has finished stabilizing...

yield return new WaitForSeconds(0.5f); // just to calm down from being so fast...

}

if(hasRemainingSpecials()){

popASpecialPiece();

yield return new WaitForSeconds(gm.gameUpdateSpeed); // wait for gravity

} else {

break;

}

}

} else { // the feature is disabled

while(gm.checkedPossibleMove == false){

// pause here till board has finished stabilizing...

yield return new WaitForSeconds(1f); // just to calm down from being so fast...

}

}

gm.gameState = JSFGameState.GameOver; // stops gameManager aswell...

validateWinLose();

}

void validateWinLose(){

int starStatus = 0; // just a little extra star status ( 3 star system game )

string starMsg = "~You Won~\n" +

"But didn't earn any star..."; // variable message changes according to star rating...

bool playerWon = true; // initial state

// check the star status...

if(gm.score > scoreMilestone3){

starStatus = 3;

starMsg = "~You Won~\n" +

"Congrats on 3 stars~!!";

} else if(gm.score > scoreMilestone2){

starStatus = 2;

starMsg = "~You Won~\n" +

"Obtained 2 stars~!";

} else if(gm.score > scoreToReach){

starStatus = 1;

starMsg = "~You Won~\n" +

"You earned 1 star!";

}

if(isScoreGame && scoreRequiredWin && starStatus == 0 ){

playerWon = false; // fail to meet minimum score...

}

if(isClearShadedGame && shadesRequiredWin && shadesLeft > 0){

playerWon = false; // fail to clear all shades

}

if(isGetTypesGame && typesRequiredWin && !collectedAll ){

playerWon = false; // fail to collect all required colors/gems

}

if(isTreasureGame && treasureRequiredWin && (numOfTreasures > treasuresCollected) ){

playerWon = false; // fail to collect all treasures

}

// game over message in the prefab

if(GameOverMessage != null){

Instantiate(GameOverMessage);

if(playerWon){ // player won...

GameObject.Find("GameOverMsg").GetComponent<TextMesh>().text = starMsg;

} else { // player lost...

GameObject.Find("GameOverMsg").GetComponent<TextMesh>().text =

"~GAME OVER~\n" +

"You failed to achieve\nthe required goals.";

}

}

}

#endregion endgame sequence

#region other functions

// function to convert remaining time to special pieces

bool convertTime(){

if((TimeGiven - timeKeeper) >= 1){

randomSpecialABoard();

timeKeeper += secondsPerSpecial; // convert every x seconds

return true;

}

return false; // no more time to convert...

}

// function to convert remaining moves to special pieces

bool convertMoves(){

if((allowedMoves - gm.moves) >= 1){

randomSpecialABoard();

allowedMoves -= movesPerSpecial; // convert every x moves

return true;

}

return false; // no more moves to convert...

}

// randomly assign a special to this board

void randomSpecialABoard(){

JSFBoard selected = getRandomBoard();

// play audio visuals

gm.audioScript.convertingSpecialFx.play();

gm.animScript.doAnim(JSFanimType.CONVERTSPEC,selected.arrayRef[0],selected.arrayRef[1]);

// get the gameobject reference

GameObject pm = gm.pieceManager;

switch(Random.Range(0,3)){

case 0:

selected.convertToSpecialNoDestroy(pm.GetComponent<JSFHorizontalPiece>(), selected.piece.slotNum ); // convert to H-type

break;

case 1:

selected.convertToSpecialNoDestroy(pm.GetComponent<JSFVerticalPiece>(), selected.piece.slotNum ); // convert to V-type

break;

case 2:

selected.convertToSpecialNoDestroy(pm.GetComponent<JSFBombPiece>(), selected.piece.slotNum ); // convert to T-type

break;

}

}

JSFBoard getRandomBoard(){ // as the title sez, get a random board that is filled...

JSFBoard selected;

List<JSFBoard> randomBoard = new List<JSFBoard>();

foreach(JSFBoard \_board in gm.board){

randomBoard.Add(\_board); // a list of all the boards in the game

}

while(randomBoard.Count > 0){ // repeat while list is not empty

selected = randomBoard[Random.Range(0,randomBoard.Count)];

if(selected.isFilled && selected.piece.pd is JSFNormalPiece) {

return selected;

}

randomBoard.Remove(selected); // remove the board from the list once checked.

}

while(true){ // contingency plan... choose a non-special powered gem

selected = gm.board[Random.Range(0,gm.boardWidth),Random.Range(0,gm.boardHeight)];

if(selected.isFilled && !selected.piece.pd.isSpecial) {

return selected;

}

}

}

// method to check if the board still has special pieces

bool hasRemainingSpecials(){

for ( int x = 0; x < gm.boardWidth; x++){

for (int y = 0; y < gm.boardHeight; y++){

if(gm.board[x,y].piece != null && gm.board[x,y].piece.pd != null &&

!(gm.board[x,y].piece.pd is JSFNormalPiece)

&& gm.board[x,y].piece.pd.isDestructible){

return true;

}

}

}

return false;

}

// method to cause a special piece to trigger it's ability

void popASpecialPiece(){

for ( int x = 0; x < gm.boardWidth; x++){

for (int y = 0; y < gm.boardHeight; y++){

if(gm.board[x,y].piece != null && gm.board[x,y].piece.pd != null &&

!(gm.board[x,y].piece.pd is JSFNormalPiece)

&& gm.board[x,y].piece.pd.isDestructible ){

gm.board[x,y].forceDestroyBox(); // force pop the special piece

gm.notifyBoardHasChanged();

return;

}

}

}

}

// function to set up the types remaining to get for this game

void setUpTypes(){

if(placeholderPanel != null){

int count = 0;

for(int x = 0; x < gm.pieceTypes.Length;x++){ // creates the visual cue on the panel

if(numToGet[x] > 0 && x < gm.NumOfActiveType){

// the visual image for player reference (e.g., red gem)

GameObject img = (GameObject) Instantiate(gm.pieceTypes[0].skin[x]);

img.transform.parent = placeholderPanel.transform;

// auto scaling feature

Bounds bounds = JSFUtils.findObjectBounds(img);

float val = 2.5f / // get the bigger size to keep ratio

Mathf.Clamp( Mathf.Max(bounds.size.x,bounds.size.y),0.0000001f,float.MaxValue);

img.transform.localScale = (new Vector3 (val, val, val )); // the final scale value

img.transform.localPosition = new Vector3 (1,-(count\*3+3),0); // position going downwards

// the text object and its position

if(textHolder) desc[x] = ((GameObject) Instantiate(textHolder)).GetComponent<TextMesh>();

desc[x].transform.parent = placeholderPanel.transform;

desc[x].transform.localPosition = new Vector3 (5,-(count\*3+3),0); // position going downwards

count++;

}

}

} else { // warning developers of missing panel reference...

Debug.LogError("Placeholder panel missing for types... unable to create." +

"Check winning conditions script again!");

}

}

public bool canSpawnTreasure(){

if( isTreasureGame && (treasuresCollected + treasureList.Count) < numOfTreasures &&

treasureList.Count < maxOnScreen){

int probability = (int) (1.0/(chanceToSpawn/100.0) );

int result = Random.Range( 0 , probability ); // random chance to spawn

if( result == 0){

return true; // spawn a treasure

}

}

return false; // cannot spawn...

}

#endregion other functions

#region important phases

// set up the variables

void Start () {

// disable those not used...

if(!isTimerGame){

if(timeLabel) timeLabel.gameObject.SetActive(false);

if(timeText) timeText.gameObject.SetActive(false);

}

if(!isMaxMovesGame){

if(movesLabel != null) movesLabel.gameObject.SetActive(false);

if(movesText != null) movesText.gameObject.SetActive(false);

}

if(!isTreasureGame){

if(treasureLabel != null) treasureLabel.gameObject.SetActive(false);

if(treasureText != null) treasureText.gameObject.SetActive(false);

}

if(!isGetTypesGame){ // game type not active... disable panel

GameObject leftPanel = GameObject.Find("CollectGamePanel"); // REVISE THE NAME if needed!

if(leftPanel != null){

leftPanel.SetActive(false); // disable this panel...

} else { // tell user the error!

Debug.LogError("you have moved/renamed the left panel for \"Get types\" game." +

"please revise Winning Conditions script!");

}

} else { // game type is active... set the stuff required!

setUpTypes();

}

StartCoroutine( updateStatus() );

StartCoroutine( routineCheck() );

if(startGameImmediately) startThisGame();

}

// function to start the timer running as well as to call GameManger's start sequence...

public void startThisGame(){

StartCoroutine( timer() );

gm.StartGame();

}

#endregion important phases

}