|  |
| --- |
| **Laboratorio 5** |
| * In questo laboratorio è stato introdotto un gioco realizzato interamente in Javascript (di Tanganelli). * Nel file .zip sono presenti diverse versioni: le prime due sono basate su un approccio semplicistico, la terza e la quarta sono realizzate adottando come approccio la programmazione ad oggetti (metodologia consigliata). * **Spiegazione del gioco**: il giocatore deve catturare più prede possibili senza essere catturato a sua volta dai nemici. Oltre ai nemici sono presenti dei *malus* per ostacolare il giocatore. I componenti presenti sono:      * + il giocatore, rappresentato con una pallina verde (che segue il nostro cursore);   + quadrati verdi, che rappresentano le prede;   + palline nere, che rappresentano nemici;   + palline bianche, che rappresentano i *malus*[[1]](#footnote-1). * **Ulteriori caratteristiche**:   + All’avvio è presente sul campo di gioco solamente il giocatore e la preda. Questo significa che non avremo nemici finchè non cattureremo la prima preda.   + Ogni volta che il giocatore cattura una preda viene generato un nemico o un *malus*.     - La probabilità che venga generato un nemico è dell’80%   + Se il giocatore viene catturato dal nemico il gioco termina stampando il punteggio ottenuto dall’utente.   + Se il giocatore viene catturato da un *malus* i nemici presenti sul campo di gioco vengono resi gradualmente invisibili per un breve periodo. Attenzione: i nemici vengono SOLO NASCOSTI, possiamo sempre toccarli.   + Il punteggio viene calcolato con le seguenti modalità:     - Si parte da zero;     - Il punteggio viene incrementato di un punto ogni volta che catturiamo una preda;     - Il punteggio viene incrementato di un ulteriore punto nel caso in cui la preda sia stata catturata nel periodo in cui i nemici sono invisibili. * **Grafica**:   + La grafica di gioco è implementata mediante codice CSS.   + Ciascun elemento presente nel campo di gioco consiste in un div inserito nel DOM.   + L’aspetto grafico degli elementi presenti all’interno del campo di gioco è implementato utilizzando la proprietà background-image. Inoltre andiamo a definire, per ciascun elemento, lunghezza e larghezza:     - Pallina verde (giocatore): 20px x 20px     - Quadrato verde (preda): 20px x 20px     - Pallina nera (nemico): 10px x 10px     - Pallina bianca (*malus*): 10px x 10px   + Lo stesso campo di gioco è un div avente dimensioni 320px x 320px. |
| **Contenuto del file game.css**  Non comprendo l’utilità del margin-top. La distanza tra top della viewport e area di gioco è determinata dal padding  body{  ~~margin-top:1%;~~  padding: 1%;  background:#FFFFE0;  }  Con margin centriamo l’area di gioco all’interno della viewport. Con cursor nascondiamo il cursore quando ci muoviamo all’interno dell’area di gioco (il nostro cursore sarà, di fatto, la pallina verde che rappresenta il giocatore). Con background e box-shadow, infine, indichiamo proprietà grafiche dell’area di gioco.  #playground{  margin:auto;  padding:0px;  cursor: none;  background:#FFD700;  box-shadow: inset 5px 5px 50px #808080;  }  /\*\*\*\*\*\*\*\*\*\*\*\* OGGETTI ALL’INTERNO DELL’AREA DI GIOCO \*\*\*\*\*\*\*\*\*\*\*\*/  #player{  width: 20px;  height: 20px;  position:absolute mi permette di alterare il normale comportamento degli elementi portandoli fuori dal normale flusso degli elementi. Definirò la loro posizione all’interno dell’area di gioco con le proprietà top, left, right, bottom.  background-image: url('./img/player.png');  position: absolute;  }  #prey{  width: 20px;  height: 20px;  background-image: url('./img/prey0.png');  position: absolute;  }  .ball{  width: 10px;  height: 10px;  /\* background-image changes according to the ball type\*/  position: absolute;  }  Definiamo le proprietà del giocatore, della preda, dei nemici e del *malus* Le proprietà di questi ultimi due sono definite insieme, rimandando la gestione dello sfondo alla proprietà background-image posta come contenuto dell’attributo style dell’elemento.  /\*\*\*\*\*\*\*\* ELEMENTI UTILIZZATI NELL’AREA CON LE INFO SUI PUNTEGGI \*\*\*\*\*\*\*\*\*/  .label {  text-align: center;  font-weight:bold;  }    .label span {  font-weight:normal;  }  /\*\*\*\*\*\*\*\* GRAFICA DEL POPUP CHE SEGNALA LA FINE DEL GIOCO \*\*\*\*\*\*\*\*\*/  /\* Definisco l’area all’interno del quale sarà presente il nostro box. Imposto con le proprietà seguenti l’estensione di quest’area su tutta la viewport (width, height, position, left e top), e al di sopra di ogni elemento presente nella pagina HTML (z-index). Con il background-color indico come colore un “grigio trasparente”. \*/  #gameoverPopup{  width: 100%;  height: 100%;  left: 0px;  top: 0px;  background-color: rgba(116,126,138,0.5);  position: fixed;  z-index: 100;  }  /\* L’unica anchor presente all’interno dell’area sarà il tasto di chiusura del box. Con le proprietà ne stabilisco la grafica (width, height, background) e la posizione (position, left, top)\*/  #gameoverPopup a{  **Dai laboratori precedenti**: *Con position:relative l’elemento viene posizionato relativamente al suo box contenitore. In questo caso il box contenitore è rappresentato dal posto che l’elemento avrebbe occupato nel normale flusso del documento.* Attenzione alla freccia arancione  width: 32px;  height: 32px;  position: relative;  display: block;  left: 95%;  top: -16px;  background: url('./img/close\_button.png') no-repeat;  z-index: 102  }  /\* Le proprietà sono le stesse (non comprendo perché ripetere le proprietà width, height, display, position, left, top e z-index). L’unica differenza sta nell’immagine (l’effetto di hovering viene dato dal cambio dell’immagine) \*/  #gameoverPopupContent a:hover{  ~~width: 32px;~~  ~~height: 32px;~~  ~~display: block;~~  ~~position: relative;~~  ~~left: 95%;~~  ~~top: -16px;~~  ~~z-index: 102;~~  background: url('./img/close\_button\_over.png') no-repeat;  }  /\* Definisco la grafica del box. Con background-color, border, border-radius e box-shadow indico colori, sfumature e bordi. Le dimensioni si adeguano alla viewport (width, height), ma non possono essere inferiori a 220px x 220px (min-width, min-height). Il posizionamento è indicato in modo tale da rendere il box centrale. \*/  #gameoverPopupContent{  width: 30%;  height: 40%;  min-width: 220px;  min-height: 220px;  position: relative;  left: 35%; /\* (35% + width/2 = 50%)\*/  top: 30%; /\* (30% + height/2 = 50%) \*/  background-color: rgba(250,250,210,1);  border: 2px solid;  border-radius: 5px;  box-shadow:0 3px 7px rgba(0,0,0,0.25);  }  /\* All’interno di #gameoverPopupContent sarà presente un unico div: quello contenente il punteggio ottenuto nella partita appena conclusa. Definisco padding, “spessore del font”, centralità e dimensione. \*/  #gameoverPopupContent div{  padding: 3pt;  font-weight: bold;  text-align: center;  font-size: x-large;  }    /\* Titolo nel box, mostrato solo se l’utente supera il punteggio massimo salvato. \*/  #highScoreText{  font-size: xx-large;  color: darkred;  } |
| **Contenuto della index.html**  <!DOCTYPE html>  <html lang="it">  <head>  <meta charset="utf-8">  <meta name = "author" content = "PWEB">  <meta name = "keywords" content = "game">  <link rel="shortcut icon" type="image/x-icon" href="./css/img/favicon.ico" />  <link rel="stylesheet" href="./css/game.css" type="text/css" media="screen">  File appena spiegato  <script type="text/javascript" src="./js/game.js"></script>  <script type="text/javascript" src="./js/ball.js"></script>  <script type="text/javascript" src="./js/sketcher.js"></script>  <script type="text/javascript" src="./js/popup.js"></script>  <title>Game</title>  Punto di partenza  </head>  <body onLoad="begin()">  <div id="playgroundWrapper">  <div id="playground" style="width:320px; height:320px; margin:0px auto"></div>  <div id="gameStat">  <div class="label">Score:  <span id="score">0</span>  </div>  Elementi già introdotti spiegando il CSS.  <div class="label">Max-Score:  <span id="max\_score">0</span>  </div>  <div class="label">Tries:  <span id="tries">1</span>  </div>  </div>  </div>  </body>  </html> |
| **Gestione di certe informazioni e contenuto del file ball.js**  **Dalle prime righe di game.js**:  // player position  var playerX = -100;  var playerY = -100;  // square position  var preyX = -100;  var preyY = -100;  // our ball object holder  var balls = new Array();   * Lo stato del giocatore è definito attraverso due variabili globali, playerX e playerY, che mantengono rispettivamente la posizione x ed y della pallina verde. * Lo stato della preda è definito attraverso due variabili globali, preyX e preyY, che mantengono rispettivamente la posizione x ed y del quadrato verde. * Lo stato dei nemici/*malus* (che a differenza dei precedenti p ossono essere molteplici, in un certo istante) è mantenuto attraverso un array di oggetti di tipo *Ball* in balls.   + L’oggetto *Ball*, la cui struttura è definita in ball.js (riportato qua per intero), presenta le seguenti proprietà   function Ball(x, y, stepX, stepY) {  this.x = x; this.y = y;  this.stepX = stepX; this.stepY = stepY;  this.type = Math.floor(Math.random() + 0.2); // 0: nemico; 1: malus  }   * + - **x** ed **y**: cordinate della palla     - **stepX**: si definisce di quanti pixel si debba spostare la pallina, sull’asse orizzontale, al passo successivo     - **stepY**: si definisce di quanti pixel si debba spostare la pallina, sull’asse verticale, al passo successivo     - **type**: definisco il tipo di palla       * 0 -> nemico       * 1-> malus   Si osservi che il *type* non lo determiniamo noi, ma viene deciso in modo randomico con funzioni matematiche. Sapendo che Math.random() mi restituisce un valore compreso tra 0 ed 1 (1 escluso) significa che ho l’80% di probabilità di non superare uno nel risultato della somma (ricordare quanto detto sulla probabilità di avere un nemico o un *malus*). |
| **Contenuto del file game.js**  *// Dimensioni dell’area di gioco*  Variabili globali utilizzate per gestire il gioco.  var PLAYGROUND\_WIDTH;  var PLAYGROUND\_HEIGHT;  var BALL\_RADIUS = 5; *// Raggio palla*  var PLAYER\_RADIUS = 10; *// Raggio giocatore*  var PREY\_HALF = 10; *// Metà lato preda*  *// Distanze (per verificare se il giocatore ha toccato qualcosa*  var MIN\_BALL\_PLAYER\_DISTANCE = 15;  var MIN\_BALL\_PLAYER\_DISTANCE\_SQUARE = 225;  var MIN\_PREY\_PLAYER\_DISTANCE = 20;  var MIN\_PREY\_PLAYER\_DISTANCE\_SQUARE = 484;  var gameTimer = null; *// Per la clearInterval*  var playground = null; *// document.getElementById(‘playground’)*  *// player position cache*  *var playerX = -100;*  *var playerY = -100;*  Cose già dette prima.  *// square position cache*  *var preyX = -100;*  *var preyY = -100;*  *// our ball object holder*  *var balls = new Array();*  *// Per lo spostamento della palla nell’area di gioco*  var NEXT\_STEP\_FACTOR = 5;  *// Gestione dei punteggi*  var playCount = 1;  var currentScore = 0;  var bestScore = 0;  *// Salvataggio dell’istante di contatto col malus*  var lastTransparencyBallTime = -1;  *// Funzione eseguita dopo il caricamento della pagina*  function begin(){  *// Recupero l’area di gioco*  playground = document.getElementById('playground');    *// Salvo in due variabili globali le dimensioni dell’area di gioco*  PLAYGROUND\_WIDTH = parseInt(playground.style.width);  PLAYGROUND\_HEIGHT = parseInt(playground.style.height);    *// Funzionalità nascosta, vedere più avanti*  playground.onclick = explode;   *// Gestisco l’esecuzione, ripetuta, della funzione clock*  *// Il gioco viene sospeso se il cursore esce dall’area di gioco, ripreso se rientriamo*  playground.onmouseenter = start;  playground.onmouseleave = pause;  *// Gestisco il movimento della pallina verde*   *(che mi rappresenta, e che sostituisce il cursore)*  playground.onmousemove = playerMoveHandler;    *// Creo la prima preda, quella che vediamo quando carichiamo il gioco*  createPrey();  }  *// Funzione eseguita ogni 20ms*  function clock() {  *// Per ogni palla presente nell’area di gioco*  for (var i = 0; i < balls.length; i++) {  moveBall(balls[i]); *// Aggiorno lo stato della palla*    *// La funzione è dichiarata nel file sketcher.js*  drawBall(i); *// Aggiorno la posizione grafica nell’area di gioco*  *// Verifico se il giocatore è stato colpito dalla pallina*  if (isBallHit(balls[i])) {  *// Se la pallina è nera (verifico il tipo) termino il gioco*  if (balls[i].type === 0) {  gameover();  }  else { *// Se non è nera è bianca, quindi rendo trasparente*  if (lastTransparencyBallTime === -1)  lastTransparencyBallTime = Date.now();  }  }  }  }  */\* Definisco una nuova posizione della preda. La determino in modo randomico in modo tale da non finire fuori dall’area di gioco. \*/*  function createPrey(){  preyX = Math.round(Math.random() \* (PLAYGROUND\_WIDTH-PREY\_HALF\*2) +   + PREY\_HALF + playground.offsetLeft);  preyY = Math.round(Math.random() \* (PLAYGROUND\_HEIGHT-PREY\_HALF\*2) +   + PREY\_HALF + playground.offsetTop);  }  */\* Creo una nuova ball calcolando in modo randomico una posizione. La formula dipende dalle dimensioni dell’area di gioco (devo definire qualcosa che sia contenuto nell’area di gioco) \*/*  function createBall() {  var x, y;  var index = balls.length;    x = Math.round(Math.random() \* (PLAYGROUND\_WIDTH-BALL\_RADIUS\*2) + BALL\_RADIUS +  + playground.offsetLeft);  y = Math.round(Math.random() \* (PLAYGROUND\_HEIGHT-BALL\_RADIUS\*2) + BALL\_RADIUS+  + playground.offsetTop);  balls.push(new Ball(x, y,  Math.random() \* NEXT\_STEP\_FACTOR - NEXT\_STEP\_FACTOR/2,  Math.random() \* NEXT\_STEP\_FACTOR - NEXT\_STEP\_FACTOR/2  ));  drawBall(index);  }  */\* Funzionalità nascosta: se clicco allontano i nemici/malus attorno a me. Più la distanza tra il nemico/malus e il giocatore è piccola, maggiore sarà la velocità guadagnata dal nemico/malus \*/*  function explode(ev) {  var x = ev.clientX;  var y = ev.clientY;  for (var i = 0; i < balls.length; i++) {  var currentBall = balls[i];  var distance = ((currentBall.x - x) \* (currentBall.x - x) + (currentBall.y - y) \*  \* (currentBall.y - y));  console.log(distance);  var distance = Math.pow(currentBall.x - x, 2) + Math.pow(currentBall.y - y, 2);  console.log(distance);  currentBall.stepX += (currentBall.x - x) / distance \* (PLAYGROUND\_WIDTH/2);  currentBall.stepY += (currentBall.y - y) / distance \* (PLAYGROUND\_HEIGHT/2);  }  }  */\* Conclusione del gioco. Blocco l’esecuzione della funzione clock, reimposto i valori di default, stampo il popup che indica il punteggio finale, rimuovo tutti gli elementi presenti nell’area di gioco, modifico il miglior punteggio se abbiamo fatto record \*/*  function gameover(){  clearInterval(gameTimer);  gameTimer = null;  lastTransparencyBallTime = -1;    *// Funzione in sketcher.js*  createPopup();  *// Funzione in sketcher.js*  removeAll();  balls = new Array();  if (bestScore < currentScore) {  bestScore = currentScore;  }  currentScore = 0;  playCount++;  *// Funzione in sketcher.js*  updateStat();  }  */\* Aggiorno la posizione del nemico/malus modificando solo lo stato del corirspettivo oggetto. Inoltre controlla che la posizione non esca dal campo di gioco (quando si trova al bordo gestisce la cosa in modo tale da avere un rimbalzo dell’elemento).*  *I ragionamenti sono molto simili a quelli visti per l’Arkanoid nello scorso laboratorio \*/*  function moveBall(ball){  if (ball.x > (PLAYGROUND\_WIDTH-BALL\_RADIUS + playground.offsetLeft)) {  ball.x = PLAYGROUND\_WIDTH-BALL\_RADIUS + playground.offsetLeft;  ball.stepX = -ball.stepX;  }  else if (ball.x < (BALL\_RADIUS + playground.offsetLeft)) {  ball.x = BALL\_RADIUS + playground.offsetLeft;  ball.stepX = -ball.stepX;  }    if (ball.y > (PLAYGROUND\_HEIGHT-BALL\_RADIUS + playground.offsetTop)) {  ball.y = PLAYGROUND\_HEIGHT-BALL\_RADIUS + playground.offsetTop;  ball.stepY = -ball.stepY;  }  else if (ball.y < (BALL\_RADIUS + playground.offsetTop)) {  ball.y = BALL\_RADIUS + playground.offsetTop;  ball.stepY = -ball.stepY;  }  ball.x += ball.stepX;  ball.y += ball.stepY;  }    function isBallHit(ball){  return (((playerX - ball.x) \* (playerX - ball.x) +  + (playerY - ball.y) \* (playerY - ball.y)) <= MIN\_BALL\_PLAYER\_DISTANCE\_SQUARE);  }    function isPreyHit(){  return !(Boolean(Math.floor(Math.abs(playerX - reyX)/MIN\_PREY\_PLAYER\_DISTANCE)+  + Math.floor(Math.abs(playerY - preyY)/MIN\_PREY\_PLAYER\_DISTANCE)))  &&  ((playerX - preyX) \* (playerX - preyX) +  + (playerY - preyY) \* (playerY - preyY)) <= MIN\_PREY\_PLAYER\_DISTANCE\_SQUARE;  }  */\* Abbiamo già detto che il cursore del mouse viene nascosto all’interno dell’area di gioco. Questa funzione viene eseguita quando ci troviamo all’interno dell’area di gioco \*/*  function playerMoveHandler(evt) {  evt.preventDefault();  *// Recupero le coordinate del mouse*  playerX = evt.clientX;  playerY = evt.clientY;  *// Verifico di non essere uscito dall’area di gioco col cursore*  *// in tal caso modifico le coordinate per riportarmi dentro*  if (playerX > (PLAYGROUND\_WIDTH-PLAYER\_RADIUS + playground.offsetLeft)) {  playerX = PLAYGROUND\_WIDTH-PLAYER\_RADIUS + playground.offsetLeft;  }  else if (playerX < (PLAYER\_RADIUS + playground.offsetLeft)) {  playerX = PLAYER\_RADIUS + playground.offsetLeft;  }    if (playerY > (PLAYGROUND\_HEIGHT-PLAYER\_RADIUS + playground.offsetTop)) {  playerY = PLAYGROUND\_HEIGHT-PLAYER\_RADIUS + playground.offsetTop;  }  else if (playerY < (PLAYER\_RADIUS + playground.offsetTop)) {  playerY = PLAYER\_RADIUS + playground.offsetTop;  }    *// Aggiorno graficamente la posizione*  drawPlayer();    *// Se la preda è stata colpita*  if (isPreyHit()) {  createPrey(); *// Creo una nuova preda*  drawPrey(); *// Inserisco la preda nell’area di gioco*  createBall(); *// Creo un nuovo nemico/malus*    *// Determino l’incremento in base alla situazione dove mi trovo*  *trasparenza dei nemici o non trasparenza?)*  currentScore += ((lastTransparencyBallTime === -1) ? 1 : 2);  *// Aggiorno le statistiche sotto l’area di gioco (funzione in sketcher.js)*  updateStat();  }  }  *// Funzione eseguita quando entro nell’area di gioco*  *// Stabilisco l’esecuzione periodica, ogni 20ms, della funzione clock (gameTimer è variabile globale)*  function start(){  drawPrey();  if (gameTimer === null)  gameTimer = setInterval(clock, 20);  }  *// Funzione eseguita quando esco dall’area di gioco*  *// Con la clearInterval blocco l’esecuzione periodica della funzione clock*  function pause(evt){  clearInterval(gameTimer);  gameTimer = null;  lastTransparencyBallTime = -1;  } |
| **Contenuto del file sketcher.js**  var PREY\_ID = 'prey';  var PLAYER\_ID = 'player';    var playerNode = null;  var preyNode = null;  var TRANSPARENCY\_PERIOD = 2000;  */\* Se la preda non è presente la creo e la inserisco nell’area di gioco. Successivamente imposto la posizione della preda nell’area di gioco (ricordiamo che la preda è unica e che cambia di posizione ogni volta la catturiamo)\*/*  function drawPrey(){  if (preyNode === null){  preyNode = document.createElement('div');  preyNode.setAttribute('id', PREY\_ID);  playground.appendChild(preyNode);  }  preyNode.style.left = (preyX-PREY\_HALF)+ 'px';  preyNode.style.top = (preyY-PREY\_HALF) + 'px';    }  */\* Gli stessi discorsi fatti prima valgono per la palla verde che rappresenta il player (fate attenzione, finchè non entriamo per la prima volta nell’area di gioco non vediamo alcuna palla verde) \*/*  function drawPlayer(){  *// Creazione della palla verde quando entriamo per la prima volta nell’area di gioco*  if (playerNode === null){  playerNode = document.createElement('div');  playerNode.setAttribute('id', PLAYER\_ID);  playground.appendChild(playerNode);  }  playerNode.style.left = (playerX-PLAYER\_RADIUS)+ 'px';  playerNode.style.top = (playerY-PLAYER\_RADIUS) + 'px';  }  */\* Aggiorno fisicamente la ball. Verifico se esiste la ball nel DOM, in caso contrario la creo e gli assegno tutte le proprietà tipiche. Si assegna anche lo sfondo (che avevamo lasciato in sospeso). L’assegnazione dello sfondo (quello da nemico o quello da malus?) dipende dal tipo deciso in modo randomico con la creazione dell’oggetto ball. \*/*  function drawBall(index){  var ballNodeId = 'ball\_' + index;  var ballNode = document.getElementById(ballNodeId);  if (ballNode === null){  ballNode = document.createElement('div');  ballNode.id = ballNodeId;  ballNode.setAttribute('class', 'ball');  ballNode.style.backgroundImage = "url('./css/img/ball" + balls[index].type + ".png')";  playground.appendChild(ballNode);  }    ballNode.style.left = (balls[index].x-BALL\_RADIUS) + 'px';  ballNode.style.top = (balls[index].y-BALL\_RADIUS) + 'px';    *// Gestisco la trasparenza dovuta al malus*  */\* Nella variabile salviamo l’eventuale data di contatto col malus. Quando il valore della variabile è diverso da -1 è certo che dobbiamo fare qualcosa. Ogni volta faccio la differenza tra ora e il contatto con il malus, dividendo per il periodo della trasparenza. Se il risultato della divisione è maggiore o uguale a 1 significa che il periodo è stato superato, e che quindi dobbiamo ritornare a situazione normale. L’opacità dell’elemento è ovviamente gestita con la proprietà CSS opacity (ricordiamo, valore tra 0 e 1 dove 0 è massima trasparenza e 0 massima opacità). Per ottenere la transizione verso la trasparenza utilizzo il risultato della divisione: abbiamo un picco di trasparenza per poi ritornare verso l’opacità. \*/*  if (lastTransparencyBallTime !== -1) {  var now = Date.now();  var alphaTimeMeasure = (now - lastTransparencyBallTime) / TRANSPARENCY\_PERIOD;  if (alphaTimeMeasure >= 1) {  lastTransparencyBallTime = -1;  alphaTimeMeasure = 1;  }  *// Che valore otteniamo quando alphaTimeMeasure = 1? (-1)\*(-1)=1 !!!!!*  ballNode.style.opacity = (1 - 2\*alphaTimeMeasure)\*(1 - 2\*alphaTimeMeasure);  }  else {  ballNode.style.opacity = 1;  }    }  */\* Aggiorno le statistiche sotto l’area di gioco. L’area di gioco è raccolta all’interno di un div e divisa in span. Modifico il firstChild per aggiornare il testo (in caso di dubbi rivedere la parte di Marcelloni dedicata). \*/*  function updateStat() {  var gameStats = document.getElementById('gameStat').getElementsByTagName('span');  gameStats[0].firstChild.nodeValue = currentScore;  gameStats[1].firstChild.nodeValue = bestScore;  gameStats[2].firstChild.nodeValue = playCount;  }  */\* Rimuovo tutti gli elementi. Pongo la diapositiva per delucidazioni rispetto a un codice sbagliato scritto da Tanganelli. \*/*    function removeAll(){  playground.innerHTML = "";  for (var i = elements.length-1; i >=0; i--) {  playground.removeChild(elements[i]);  }  Se si mantiene questa cosa non rimuoveremo tutti gli elementi presenti nel campo di gioco.  playerNode = null;  preyNode = null;  } |

|  |
| --- |
| **Contenuto di popup.js**  *// Identificativi (valore dell’attributo ID) degli elementi del popup*  GAMEOVER\_POPUP\_ID = 'gameoverPopup';  GAMEOVER\_POPUP\_CONTENT\_ID= 'gameoverPopupContent' ;  *// Creo un div che conterrà l’avviso di nuovo record punteggio.*  function createNewBestScoreLabel(){  var newHighScore = document.createElement('div');  newHighScore.setAttribute('id', 'highScoreText');  var newHighScoreText = document.createTextNode('NEW HIGH SCORE!!!');  newHighScore.appendChild(newHighScoreText);  return newHighScore;  }  */\* Decido il contenuto in base al punteggio, stampo un messaggio più evidente se il giocatore ha battuto il precedente record (questo messaggio più evidente è gestito da un’altra funzione). \*/*  function createScoreLabel(){  var scorePopup = document.createElement('div');  if (currentScore > bestScore)  scorePopup.appendChild(**createNewBestScoreLabel()**);    var textScorePopup = document.createTextNode('Your score is: ' + currentScore);  scorePopup.appendChild(textScorePopup);  return scorePopup;  }  */\* Creo il bottone di chiusura. Le proprietà grafiche sono già state attribuite mediante CSS, mi limito a indicare l’esecuzione di una funzione in caso di click. \*/*  function createCloseButtonPopup(){  var closeButtonPopup = document.createElement("a");  closeButtonPopup.setAttribute('onClick', 'closePopup()');  return closeButtonPopup;  }  */\* Parte tutto da qua, la funzione viene chiamata in caso di gameover nell’omonima funzione già introdotta. \*/*  function **createPopup()**{  var gameoverPopup = document.getElementById(GAMEOVER\_POPUP\_ID);  if (gameoverPopup !== null)  return;    // Creo il popup  var gameoverPopup = document.createElement('div');  gameoverPopup.setAttribute('id', GAMEOVER\_POPUP\_ID);    // Creo il div dove metteremo il contenuto del popup  var content = document.createElement('div');  content.setAttribute('id', GAMEOVER\_POPUP\_CONTENT\_ID);    // Individuo il contenuto  content.appendChild(**createCloseButtonPopup()**);  content.appendChild(**createScoreLabel()**);    // Aggiungo quanto trovato al primo div creato  gameoverPopup.appendChild(content);    // Aggiungo tutto al documento  document.body.appendChild(gameoverPopup);  }  */\* Funzione eseguita quando premiamo l’anchor di chiusura del popup \*/*  function closePopup(){  var gameoverPopup = document.getElementById(GAMEOVER\_POPUP\_ID);  if (gameoverPopup === null)  return;  document.body.removeChild(gameoverPopup);  } |
| **Seconda versione**   * A questo punto il prof.Tesconi ha lasciato una ventina di minuti per realizzare nuove funzionalità all’interno del codice appena visto. * Le nuove funzionalità poste nella seconda versione sono le seguenti:      * **Differenze in index.html**:   + Introduzione del seguente elemento in prossimità dell’area di gioco   <div id="specialAbilities"></div>   * **Differenze in game.js**:   + Inserimento di una nuova variabile   var preyType = -1;  che consente la gestione del tipo della nuova preda   * + Inserimento di una nuova variabile   var availableExplodeAbility = MAX\_EXPLODE\_COUNT; con cui ci ricordiamo il numero di click rimasti (utilizzo dell’abilità explode)   * + Nuova funzione   function createSpecialAbilityImages(playgroundWrapper){  var specialAbilitiesElement = playgroundWrapper.childNodes[1];  for(var i = 0; i < MAX\_EXPLODE\_COUNT; i++){  var img = new Image();  img.src = './css/img/explosion.png';  img.alt = 'special ability number ' + (i+1);  specialAbilitiesElement.appendChild(img);  }  } con cui inseriamo tre simboli che rappresentano il numero di utilizzi dell’abilità explode    La funzione viene chiamata all’interno di begin  createSpecialAbilityImages(document.getElementById('playgroundWrapper'));   * + ­Gestione del tipo della preda in createPrey:   function createPrey(){  […]  *// Attenzione alla probabilità richiesta, del 5%*  preyType = Math.floor(Math.random() + **0.05**);  if (preyType === 1)  ballPreyTimeout = setTimeout(removePrey, 2000);  }  eseguo dopo due secondi la seguente funzione  function removePrey(){  createPrey();  drawPrey();  }   * + Modifiche nella funzione explode per la gestione dell’abilità omonima   function explode(ev) {  if (availableExplodeAbility <= 0)  return;    […]    availableExplodeAbility--;  updateSpecialAbilities();  }  Fermo subito l’esecuzione della funzione se ho già consumato il numero di possibilità disponibili. Se utilizzo una possibilità, quindi vado avanti, decremento il numero di possibilità rimaste ed eseguo la funzione updateSpecialAbilities  function updateSpecialAbilities(){  var imageElements = document.getElementById('specialAbilities').getElementsByTagName('img');  var i = 0;  for (; i < availableExplodeAbility; i++)  imageElements[i].style.opacity = 1;    for (; i < imageElements.length; i++)  imageElements[i].style.opacity = 0.3;    }  Col primo for rendo opache tante immagini quante le possibilità rimaste, col secondo for (che riprende la variabile i incrementata nel primo for) rendiamo quasi trasparenti le immagini rimanenti (le possibilità già consumate)   * + Modifiche nella funzione playerMoveHandler   function playerMoveHandler(evt) {  […]  if (isPreyHit()) {  *// Tutto normale se il tipo di preda è quello classico*  if (preyType === 0)  currentScore += lastTransparencyBallTime === -1 ? 1 : 2;  else{  *// Resetto il timeout, quello che mi fa sparire la preda dopo due secondi*  clearTimeout(ballPreyTimeout);  ballPreyTimeout = null;   *// Incremento di 5 lo score*  currentScore += 5;  *// Se non ho già il massimo di possibilità per l’abilità*   *explode incremento*  if (availableExplodeAbility < MAX\_EXPLODE\_COUNT)  availableExplodeAbility++;  *// Solita funzione con cui aggiorniamo le immagini   poste in prossimità del playground*  updateSpecialAbilities();  }  […]  }  } |
| **Versione 3**  Per la parte rimanente mi limiterò a proporre, nelle pagine successive, le diapositive utilizzate. Il prof.Tesconi si è dedicato soprattutto alla lettura delle diapositive, che introducono alcuni concetti teorici molto importanti.  Si osservi, in particolare:   * L’adozione del localStorage per gestire il punteggio massimo (secondo le regole indicate) * La possibilità, grazie alla programmazione ad oggetti, di poter gestire più aree di gioco all’interno della stessa pagina (senza programmazione ad oggetti questa cosa sarebbe IMPOSSIBILE) * L’adozione del modello architetturale MVC (*ModelView-Controller*) nella programmazione ad oggetti. * Il significato dell’attributo this, concepito in Javascript in modo diverso da come siamo abituati (ripensiamo al C++). |

1. Il malus può sembrare un vantaggio per il giocatore, ma in realtà non lo è. [↑](#footnote-ref-1)