U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

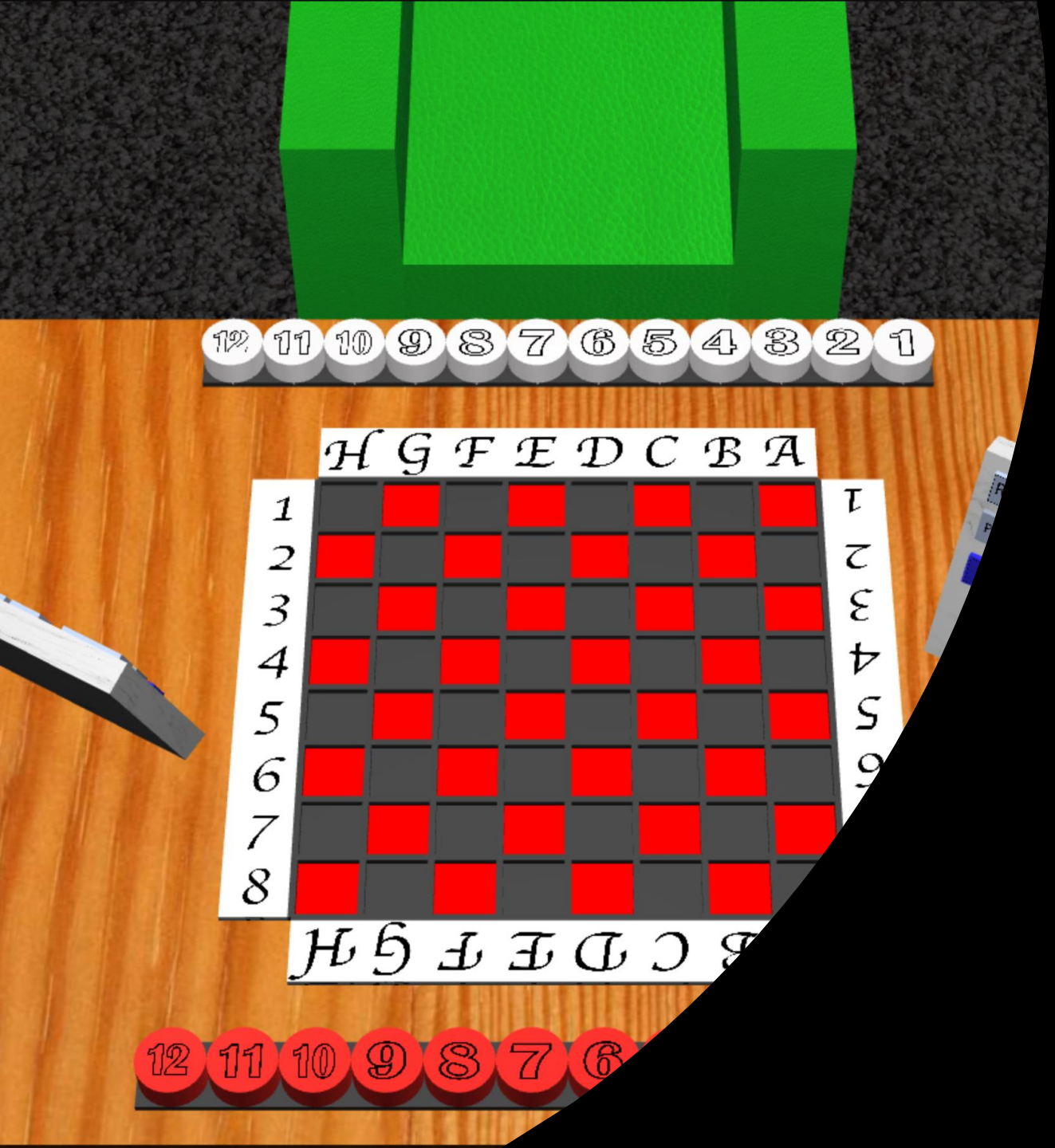
Interactive Graphics Systems

Thoughts on game development

v1.0 20221205 draft

Alexandre Valle | DEI | FEUP

Pictures: two details of HCI from movie T*he Dark Knight (credits BLIND, Ltd.)*



Thoughts on game development

1.Game requisites

Game requisites: lighting

- Add the appropriate light sources to brighten the scene for a realistic look.

- Solution: use the scene file and carefully configure the lights!

Game requisites: scene

- Develop a main board suitable for the game you choose.
- Auxiliary boards may be required!
- Implement a set of predefined environments, allowing to choose from a variety of themes.

Game requisites: scene

- **Solution**:
  - A scene file contains multiple themes:
    - One single large scene file:
      - “high” complexity in the file: lots of lines….
      - Requires changes in the parser to read only some sections.
  - Multiple scene files, one per theme:
    - Manageable complexity per file
    - No need to add additional semantic for the parser to read sections.

Game requisites: scene

- The scene file could be extended to have new “semantic” primitives. One or more of the following:

<mainboard x1 y1 x2 y2>
<auxiliarboard1 x1 y1 x2 y2>
<auxiliarboard…>
<auxiliarboardn>
<piecetype1>
<piecetype...>
<piecetypen>
<tiletype1>
<tiletype...>
<tiletypen>

Game requisites: game sequence

- Game turn requisites from:
  - Pieces that can be removed or inserted during game play.
  - Game users: with a mouse click selects the **piece** to move; a new click on a **destination tile** of the board designates the target position;

Game requisites: animation

- Pieces can be removed or inserted during game play.
- Pieces should NOT simply appear or disappear.
- A piece must move in an animation and do not collide/traverse other pieces.
- Consider an auxiliary board to keep pieces coming out of the game.

Requisites: UI game features

Build an interface to include options such as, for example:

- Undo, i.e. possibility to undo last or last moves.
- Rotate the camera between predefined views (at least two).
- Marker record the game results. Time clock.
- Other…

UI game features

- UI overlaying WEBGL canvas

UI game features

- UI inside WEBGL canvas (pickable objects)

Undo and game move

Keep the game sequence:

- List of game moves (alternative 1)
  - By traversing the sequence of moves you get a gameboard state from the beginning and up to any point of the game.
- List of gameboard states (alternative 2)
  - By “subtracting” two consecutive gameboard states you get the game move.
- Gameboard state and game move (alternative 3)
  - All data is stored to render game move and gameboard states.

Undo and game play

- Undo:
  - Remove item from the end of list (one item = one move)
  - Render de gameboard state

- Game play:
  - Assume the first gameboard state
  - For each game move:
    - animate the game move
    - Consider it current gameboard state

2.Game states

(generalization)

Game states (assuming a chess game)

- **Menu ->** show menu and handle settings.
- **Load scenario** -> (keep game state), load file, render scene, board, pieces, etc.
- **Next turn**
  - Human 1? Wait for **pick** piece.
  - Human 2? Wait for **pick** piece.
- **Render possible moves**-> after previous state, render possible target tiles
  - Human 1? render and move to next state.
  - Human 2? render and move to next state.

Game states (assuming a chess game)

- **Destination piece/tile selection** -> after previous state**,**
  - Human 1? Wait for **pick** destination tile/piece.
  - Human 2? Wait for **pick** destination tile/piece.
- **Movement animation** -> after previous state, selection object is moved based on some defined animation f(t).
- **Has game ended?** -> after previous state, evaluate if **End Game** or **Next turn**.
- **End game -> display winner and go to menu**

- Game states should be managed by GameOrchestrator (further in presentation)

Interrupting game states

- The following state may interrupt previous game states:
    - **Undo ->** undo the last game move. Updates game sequence and turn.
    - **Movie ->** keep current game state. Renders all the game movements (should use the same animation features used for **movement animation**). After, return to current game state.
    - **Load scennario ->** keep game state. Load file render scene, board, pieces, etc. Return to current game state.
- The rule of thumb is that by the end of each of these interrupting states the game is returned to a previous “stable” game state.
- Interrupting states should be managed by **GameOrchestrator** (further in presentation)

4.Concept classes

(depends on game)

Piece

- Game element that occupies tiles
- Class MyPiece
- Attributes: a type, a geometry, a pointer to a tile object (if a piece is placed on the gameboard/auxiliary board)
- Methods:
    - get/set type
    - Display the piece (render)

Gameboard tile

- Unitary element that creates the main game board and auxiliary board spaces.
- Class MyTile
- Atrribute: pointer to main game board, pointer to piece (if a piece occupies the tile), coordinates in the board
- Methods:
    - Set/unset piece on tile
    - Get piece using tile
    - Get/set board
    - Display the tile (render)

Gameboard

- Stores the set of tiles that composes the entire board
- Class MyGameBoard
- Methods:
  - Create a *gameboard instance*
  - *Add piece to a given tile*
  - *Remove piece from a given tile*
  - *Get piece on a given tile*
  - *Get tile given a piece*
  - *Get tile by board coordinate system (A..H;1..8 on chess or 0..7;0..7)*
  - Move piece (piece, starting tile, destination tile)
  - Display the gameboard (render). Calls display on each tile (which by its own turn calls display of the piece in the tile, if any).

Game move

- Stores a game move
- Class MyGameMove
- *Has:*
  - Pointer to moved piece (MyPiece)
  - Pointer to origin tile (MyTile)
  - Pointer to destination tile (MyTile)
  - Gameboard state before the move
- Methods:
  - Animate

Game sequence

- Stores the sequence of game moves (MyGameMove objects):
- Class MyGameSequence
- Methods:
  - Add a game move
  - Manage *undo*
  - Feeds move replay

Animator

- Manages the animation of an entire game sequence
- Class MyAnimator
- *Has:*
  - Pointer to the orchestrator
  - Pointer to the game sequence
- Methods:
  - reset
  - start
  - update(time)
  - Display. Optionally can look at the orchestrator to stop current animation.

Game orchestration

- Manages the entire game:
  - Load of new scenes
  - Manage gameplay (game states and interrupting game states)
  - Manages undo
  - Manages movie play
  - Manage object selection

  Class MyGameOrchestrator

Game orchestration

```
class MyGameOrchestrator
    …
    this.gameSequence = new MyGameSequence(…);
    this.animator = new MyAnimator(…);
    this.gameboard = new MyGameboard(…);
    this.theme = new MyScenegraph(…);
```

… = parameters are required.

Game orchestration

```
update(time) {
     this.animator.update(time);
}

display() {
     …
     this.theme.display();
     this.gameboard.display();
     this.animator.display();
     …
}
```

XMLScene

```
class XMLScene {
        update(time) {
                …
                this.gameOrchestrator.update();
        }
        …
        display () {
                this.gameOrchestrator.orchestrate();
                // general display
                this.gl.viewport(0, 0, this.gl.canvas.width, this.gl.canvas.height);
                this.gl.clear(this.gl.COLOR_BUFFER_BIT | this.gl.DEPTH_BUFFER_BIT);
                …
                this.gameOrchestrator.display();
        }
}
```

3.Object selection

(uses WEBCGF picking feature)

Pick support in XMLScene

```
init(application) {
        super.init(application);
        …
        this.setUpdatePeriod(10);
        this.setPickEnabled(true); // false to disable pick feature.
                                   // Some game states do not require pick.
}
display(){
        this.gameOrchestrator.managePick(this.pickMode, this.pickResults);
        this.clearPickRegistration();
        …
}
```

(Pick support for classes that contain selectable geometry)

```
class MyPiece  /* could be some other class */ {
   display() {
        if (this.selectable)
           this.orchestrator.getScene().registerForPick(this.uniqueId, this);
        // Now call all the game objects/components/primitives display
        // method that should be selectable and recognized
        // with this uniqueId

        // clear the currently registered id and associated object
        if (this.selectable)
           this.orchestrator.getScene().clearPickRegistration();
}
```

NOTES:

- the display method is called by the display method hierarchy starting on XMLScene > My game orchestrator > etc…

- **uniqueId** should be unique and previously provided by gameOrchestrator

MyGameOrchestrator

```
managePick(mode, results) {
   if (mode == false /* && some other game conditions */)
      if (results != null && results.length > 0) { // any results?
         for (var i=0; i< results.length; i++) {
            var obj = pickResults[i][0]; // get object from result
            if (obj) { // exists?
               var uniqueId = pickResults[i][1] // get id
               this.OnObjectSelected(obj, uniqueId);
            }
         }
         // clear results
         pickResults.splice(0, pickResults.length);
      }
   }
}
```

MyGameOrchestrator

```
onObjectSelected(obj, id) {
    if(obj instanceof MyPiece){
        // do something with id knowing it is a piece
    }
    else
    if(obj instanceof MyTile){
        // do something with id knowing it is a tile
    }
    else {
        // error ?
    }
}
```

End

Alexandre Valle
alexandre.valle@fe.up.pt