



Thoughts on LAIG work 3 game development

Alexandre Valle de Carvalho

v20191203

1.Game requisites

Game requisites: lighting

- Add the appropriate light sources to brighten the scene for a realistic look.
- **Solution:** carefully configure scene file!

Game requisites: scene

- Develop a 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

- Scene file could be extended with new primitives.
- One or more of the following:

<gameboard 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.
 - Human: with a mouse click selects the **piece** to move; a new click on a **tile** of the board designates the target position;
 - Computer: also **piece** + **destination tile**.

Game requisites: animation

- Pieces can be removed or inserted during game play.
- Parts should NOT simply appear or disappear.
- A piece must move in an animation, possibly in an arc, so as not to cross other pieces.
- Consider an auxiliary board (frame) to support these pieces from / where the pieces move.

Requisites: UI game features

- Build an interface using WebCGF's GUI options, or by creating selectable objects, and that include (at least) the following options:
- Difficulty level.
- Type of game (Human / Machine, H / H, M / M).
- 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.

UI game features

- UI overlaying WebGL canvas



UI game features

- UI inside WEBGL canvas (pickable objects)



Undo and game move

- Store the game sequence:
 - List of game moves
 - By traversing the sequence of moves you get a gameboard state.
 - Gameboard state
 - By “subtracting” two consecutive gameboard states you get the game move.
- Gameboard state and game move
 - 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? **pick** piece or tile.
 - Prolog ? [**Request(s) to prolog**] -> get piece/tile, possible moves and destination piece/tile.
- **Render possible moves**-> based on previous render possible target tiles
 - Human? render and move to next state.
 - Prolog ? render and wait a couple of seconds...

Game states (assuming a chess game)

- **Destination piece/tile selection:**
 - Human ? **pick** destination tile/piece.
 - Prolog ? render destination piece/tile.
- **Movement animation** -> selection is moved with based on some animation $f(t)$.
- **Evaluate game end** -> [**request to prolog**] and evaluate game end or **Next turn**.
- **End game** -> display winner and goto 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 movement. Updates turn.
 - **Movie** -> keep game state. Renders all the game movements (should use the same animation features used for **movement animation**).
 - **Load scenario** -> keep game state. Load file render scene, board, pieces, etc.
- The rule of thumb is that by the end of each of the previous states the game is returned to the previous “stable” state.
- Interrupting states should be managed by GameOrchestrator (further in presentation)

4. Concept classes

(depend on game)

Piece

- Game element that occupies tiles
- Class MyPiece
- Piece can hold several piece types
- Has pointer to holding tile (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 gameboard and auxiliary board spaces.
- Class MyTile
- Has pointer to gameboard and pointer to piece (if a piece occupies tile)
- Methods:
 - Set/unset piece on tile
 - Get piece using tile
 - Display the tile (render)

Gameboard

- Stores the set of tiles that composes the entire game 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 of tiles and of pieces.

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 (MyGameboard representation)
- Methods:
 - Animate

Game sequence

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

Animator

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

Game orchestration

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

Game orchestration

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

... = 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();  
    }  
}
```

Prolog interface

Encapsulates and manages prolog communications

Class MyPrologInterface

Methods:

- all that are necessary to request data to prolog
- all that are necessary to parse responses and accommodate in game data structures (centralized in GameOrchestrator)

Prolog interface

Request send example:

```
let requestString = 'playFieldsOfAction('+list+')';
let request = new MyXMLHttpRequest(this);

request.addEventListener("load", this.parseStartPrologReply);
request.addEventListener("error", this.startPrologGameError);

request.open('GET', 'http://localhost:'+PORT+'/'+requestString, true);

request.setRequestHeader("Content-type", "application/x-www-form-urlencoded; charset=UTF-8");
request.send();
```

Prolog interface

- Process reply:

```
parseStartPrologReply() {  
  
    if (this.status === 400) {  
        console.log("ERROR");  
        return;  
    }  
    // the answer here is: [Board,CurrentPlayer,WhiteScore,BlackScore]  
  
    let responseArray = textStringToArray(this.responseText,true);  
    // do something with responseArray[0];  
    // do something with responseArray[1];  
    // do something with responseArray[2];  
    // do something with responseArray[3];  
}
```

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 de Carvalho
alexandre.valle@fe.up.pt