GRIDLang

Grid Based Game Programming Language
PLT Spring 2017
Team

Player Akshay_Nagpal,
Dhruv_Shekhawat,
Parth_Panchmatia,
Sagar_Damani ;
Main Goals

• Design games in an intuitive and expressive manner

• Quickly prototype grid-based games and get a programmatic view

• Simplify the process of:
  • defining rules for a game
  • grid creation and manipulation
  • in-built language components that enable programmer to express more with less lines of code
Core Features

- Strongly typed
- Move Driven
- Structs, Pointers, Arrays(1D & 2D)
- Standard Library
Initialize Grid

Grid_Init<7, 7>;
Creating Player and item structs

Player
{
    Piece horse h1,h2,h3;
    int score;
}
Piece horse
{
    int value;
}
Adding to Grid

Player p1;

int setup(){
    p1.h1.displayString = "h1";
    p1.h2.displayString = "h2";
    p1.h3.displayString = "h3";
    Grid<3,6> <-- p1.h1;
    Grid<3,2> <-- p1.h2;
    Grid<5,2> <-- p1.h3;
    return 0;
}

printGrid();
## Grid Initialization

<table>
<thead>
<tr>
<th>NULL</th>
<th>NULL</th>
<th>NULL</th>
</tr>
</thead>
<tbody>
<tr>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
</tbody>
</table>
GenericPiece from MiniChess

Piece GenericPiece
{
    Piece King* King_node;
    Piece Pawn* Pawn_node;
    Piece Bishop* Bishop_node;
    int x, y;
    Piece GenericPiece* next;
    string nametag, typetag;
    Player* owner;
}
Piece* horse h_node;
Piece* bishop b_node = b1;
typetag = “bishop”
nametag = “b1”
owner  = “black”
Piece* horse h_node = h1;
Piece* bishop b_node = b1;
typetag = “bishop”
nametag = “b1”
owner = “black”

next
Control Flow

- gameLoop
  - moveOnGrid
    - rule
      - true: Move Piece
      - false: colocation
        - true: checkGameEnd
          - false: End
          - true: End
Mini Chess
Bishop Rule - Check if Move is Diagonal

```
if(abs(dst_x - src_x) == abs(dst_y - src_y))
```
Bishop Rule – Check if Diagonal is Blocked

```c
if (traverse(src_x, src_y, dst_x, dst_y) == 1) {
    return 0;
}
```

```
<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>B-Knight</td>
<td>B-Bishop</td>
<td>B-Rook</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>B-King</td>
<td></td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>B-Pawn</td>
<td></td>
<td></td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>W-Bishop</td>
<td>W-Pawn</td>
<td></td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>W-Rook</td>
<td></td>
<td></td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

White
Enter source x:3
Enter source y:2
Enter destination x:1
Enter destination y:0
Invalid move
```
Colocation

int colocation(int x, int y, Piece GenericPiece* i1, Piece GenericPiece* i2)
{
    deleteFromGrid(x, y, i2->nametag);
    return 0;
}
checkGameEnd (Snakes and Ladders)

```c
int checkGameEnd()
{
    Piece Token *t;
    Piece GenericPiece *token;
    t = getCurrentPlayer();
    token = getPieceFromGrid(t.displayString);
    if (token.x == 0 && token.y == 5){
        printGrid();
        print("Winner is: ");
        print(t.displayString);
        return 1;
    }
    return 0;
}
```

```
0 1 2 3 4 5
L2-Top L1-Top | L2-Bottom S1-Head
| S2-Tail | L1-Bottom | S2-Head
| P2      | S1-Head   | P1
```

Winner is: P1
```
Lessons Learned

- Have a concrete plan of what your language does.
- Team matters a lot. Choose team members based on their ability to learn.
- Two heads are better than one.