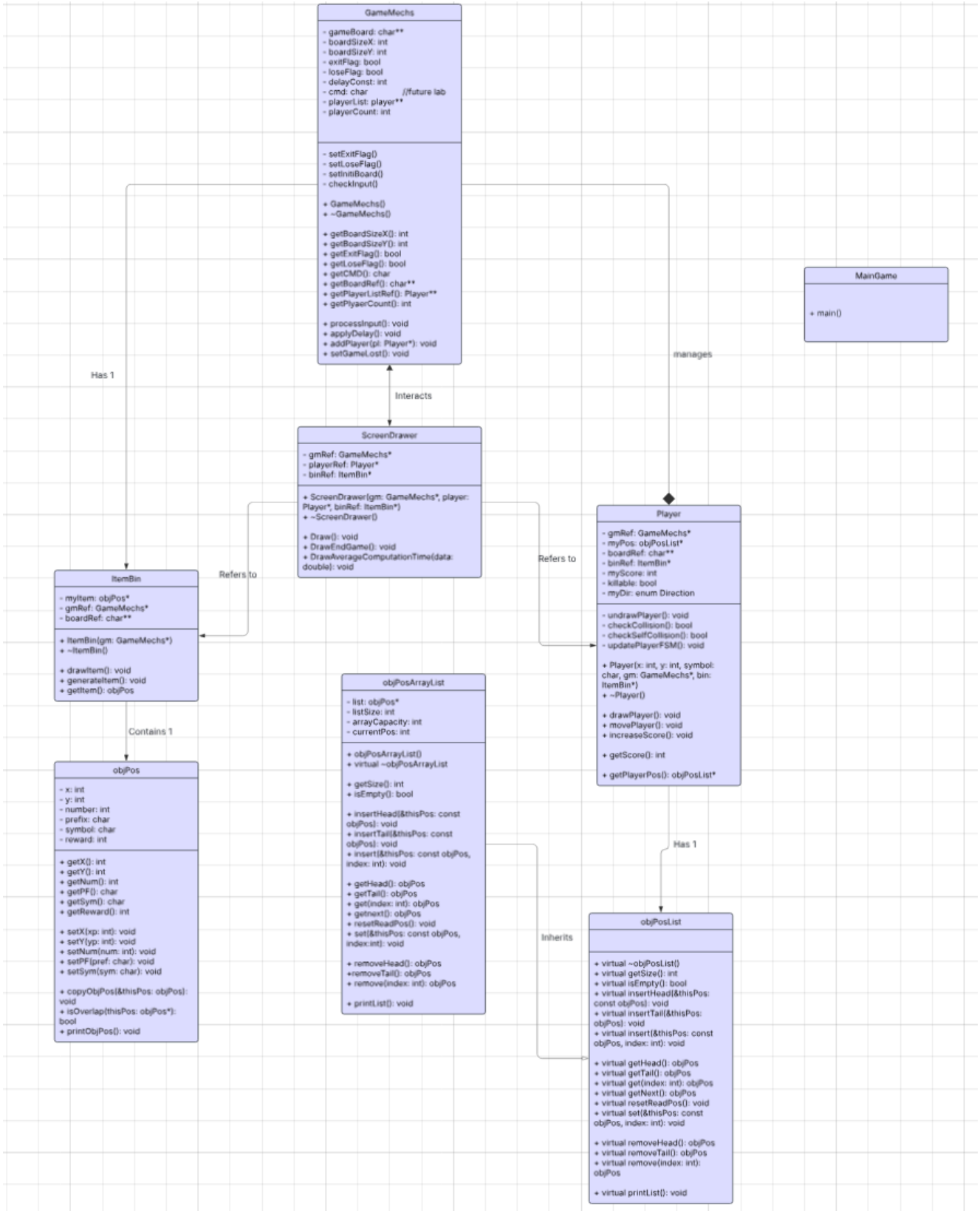
Compeng 2SI4 Lab 1

Joshua Obljubek-Thomas

Obljubej, 400506256


GameMechs
- gameBoard: char**
- boardSizeX: int
- boardSizeY: int
- exitFlag: bool
- loseFlag: bool
- delayConst: int
- cmd: char //future lab
- playerList: player**
- playerCount: int
- setExitFlag()
- setLoseFlag()
- setInitBoard()
- checkInput()
+ GameMechs()
+ ~GameMechs()
+ getBoardSizeX(): int
+ getBoardSizeY(): int
+ getExitFlag(): bool
+ getLoseFlag(): bool
+ getCMD(): char
+ getBoardRef(): char**
+ getPlayerListRef(): Player**
+ getPlyaerCount(): int
+ processInput(): void
+ applyDelay(): void
+ addPlayer(pl: Player*): void
+ setGameLost(): void
MainGame
+ main()
Has 1
Interacts
manages
ScreenDrawer
- gmRef: GameMechs*
- playerRef: Player*
- binRef: ItemBin*
+ ScreenDrawer(gm: GameMechs*, player: Player*, binRef: ItemBin*)
+ ~ScreenDrawer()
+ Draw(): void
+ DrawEndGame(): void
+ DrawAverageComputationTime(data: double): void
Refers to
Refers to
Player
- gmRef: GameMechs*
- myPos: objPosList*
- boardRef: char**
- binRef: ItemBin*
- myScore: int
- killable: bool
- myDir: enum Direction
- undrawPlayer(): void
- checkCollision(): bool
- checkSelfCollision(): bool
- updatePlayerFSM(): void
+ Player(x: int, y: int, symbol: char, gm: GameMechs*, bin: ItemBin*)
+ ~Player()
+ drawPlayer(): void
+ movePlayer(): void
+ increaseScore(): void
+ getScore(): int
+ getPlayerPos(): objPosList*
ItemBin
- myItem: objPos*
- gmRef: GameMechs*
- boardRef: char**
+ ItemBin(gm: GameMechs*)
+ ~ItemBin()
+ drawItem(): void
+ generateItem(): void
+ getItem(): objPos
Contains 1
objPos
- x: int
- y: int
- number: int
- prefix: char
- symbol: char
- reward: int
+ getX(): int
+ getY(): int
+ getNum(): int
+ getPF(): char
+ getSym(): char
+ getReward(): int
+ setX(xp: int): void
+ setY(yp: int): void
+ setNum(num: int): void
+ setPF(pref: char): void
+ setSym(sym: char): void
+ copyObjPos(&thisPos: objPos): void
+ isOverlap(thisPos: objPos*): bool
+ printObjPos(): void
objPosArrayList
- list: objPos*
- listSize: int
- arrayCapacity: int
- currentPos: int
+ objPosArrayList()
+ virtual ~objPosArrayList
+ getSize(): int
+ isEmpty(): bool
+ insertHead(&thisPos: const objPos): void
+ insertTail(&thisPos: const objPos): void
+ insert(&thisPos: const objPos, index: int): void
+ getHead(): objPos
+ getTail(): objPos
+ get(index: int): objPos
+ getnext(): objPos
+ resetReadPos(): void
+ set(&thisPos: const objPos, index:int): void
+ removeHead(): objPos
+removeTail(): objPos
+ remove(index: int): objPos
+ printList(): void
Has 1
Inherits
objPosList
+ virtual ~objPosList()
+ virtual getSize(): int
+ virtual isEmpty(): bool
+ virtual insertHead(&thisPos: const objPos): void
+ virtual insertTail(&thisPos: objPos): void
+ virtual insert(&thisPos: const objPos, index: int): void
+ virtual getHead(): objPos
+ virtual getTail(): objPos
+ virtual get(index: int): objPos
+ virtual getNext(): objPos
+ virtual resetReadPos(): void
+ virtual set(&thisPos: const objPos, index: int): void
+ virtual removeHead(): objPos
+ virtual removeTail(): objPos
+ virtual remove(index: int): objPos
+ virtual printList(): void

Note: Only the necessary lines are present to show the time complexities

a. **$\Theta(n^2)$** due to the nested for loop each having n time complexity → n*n = n^2

```
void GameMechs::setInitBoard(){  // private helper function
    for(int i = 0; i < boardSizeY; i++){  //time complexity n
        for(int j = 0; j < boardSizeX; j++){  //time complexity n
        }
    }
}
```

b. **$\Theta_{best}(1)$** due to the first if statement, **$\Theta_{worst}(n)$** → for loop running listSize/n times, our upper bound is just **O(n)**. Little o is any complexity greater than **Θ(n)** → **o(nlogn)**

```
void objPosArrayList::insertHead(const objPos &thisPos){
    if(listSize == arrayCapacity) return;  //lower bound

    // The rest will be our estimation of Theta worst case
    for(int i = listSize; i > 0; i--) {    //upper bound
    }
}
```

c. **Θ(1)** since linear complexity

```
objPos objPosArrayList::removeTail(){
    return list[listSize];
 }
```

d. **$\Theta_{best}(1)$** due to the first if statement, **$\Theta_{worst}(n)$** → for loop running listSize/n times, our upper bound is just **O(n)**. Little o is any complexity greater than **Θ(n)** → **o(nlogn)**

```
void objPosArrayList::insert(const objPos &thisPos, int index){
    if(listSize == arrayCapacity) return;   //best case (lower bound
      // The rest will be our estimation of Theta worst case
    for(int i = listSize; i > index; i--){  //worst case (upper bound)
    }
}
```

e. Both **Θ(n)**

```
void Player::drawPlayer()  {
    for(int i = 0; i < scanSize; i++{ //time complexity of n
    }
}


void Player::undrawPlayer(){  // private helper function
```

```
    for(int i = 0; i < scanSize; i++){  //time complexity of n
    }
}
```

f. In the function we see various lines with linear, n, and n^2 complexity, thus we take the largest and get **$\Theta(n^2)$**

```
void ScreenDrawer::Draw() const{
    MacUILib_clearScreen();//linear

    binRef->drawItem();     //time complexity of n
    playerRef->drawPlayer();//time complexity of n

    char** drawTarget = gmRef->getBoardRef();
    objPos target = binRef->getItem();

    for(int i = 0; i < gmRef->getBoardSizeY(); i++){//time complexity of n
        for(int j = 0; j < gmRef->getBoardSizeX(); j++)//n time{
            //n * n = n^2
        }
    }
}
```

g. Theta not applicable since we cannot get the upper bound and the lower bound to match due to the return statement in the for loop. This means we have will have a tight upper bound of **O(n)** and a lower bound of **Ω(1).** We also have **$\Theta_{worst}(n)$** and **$\Theta_{best}(1)$**, **o(nlogn)**.

```
bool Player::checkSelfCollision(){
    int length = myPos->getSize();//linear
    if(length < 4) return false; //best case

    for(int i = 1; i < length; i++){//worst case the function never
returns → O(n)for upper bound
        if(headPos.isOverlap(&tempPos)){ //best case is this returns on
first run through → Ω(1)
            return true;
        }
    }
    return false;
}
```

h. Due to the early if statement we have **$\Theta_{best}(1)$.** If that if statement fails we have **$\Theta_{worst}(n)$** due to the undrawPlayer function call.

```
void Player::movePlayer(){
    updatePlayerFSM();  //linear
    if(myDir == STOP) return;

    undrawPlayer();     //time complexity n

    switch(myDir){      //linear time complexity
        case UP:
            if(--inY < 1)
                inY = gmRef->getBoardSizeY() - 2;
            break;

        case DOWN:
            if(++inY > (gmRef->getBoardSizeY() - 2))
                inY = 1;
            break;

        case LEFT:
            if(--inX < 1)
                inX = gmRef->getBoardSizeX() - 2;
            break;

        case RIGHT:
            if(++inX > (gmRef->getBoardSizeX() - 2))
                inX = 1;
            break;

        default:
            break;
    }
}
```

i. Due to the do while loop we cannot assign an upper bound of big O since it could run forever. Due to multiple variables of n = playerSize and m = xsize,ysize. We can assign minimum time complexity of $\mathbf{\Omega(m^2) + \Omega(n)}$.

```
void ItemBin::generateItem(){
    for(int i = 0; i < xsize; i++){//time complexity m
        for(int j = 0; j < ysize; j++){//time complexity m
            //time complexity always m^2
        }
```

```
    }

    for(int i = 0; i < playerLength; i++){ //time complexity n
    }

    Do    //no upper bound for the time complexity
    {
        randCandidateX = rand() % (gmRef->getBoardSizeX() - 4) + 2;
        randCandidateY = rand() % (gmRef->getBoardSizeY() - 4) + 2;
    } while(bitVec[randCandidateX][randCandidateY] != 0);

    for(int i = 0; i < xsize; i++)// n time complexity
        delete[] bitVec[i];
    delete[] bitVec;
}
```