Bombin' Man

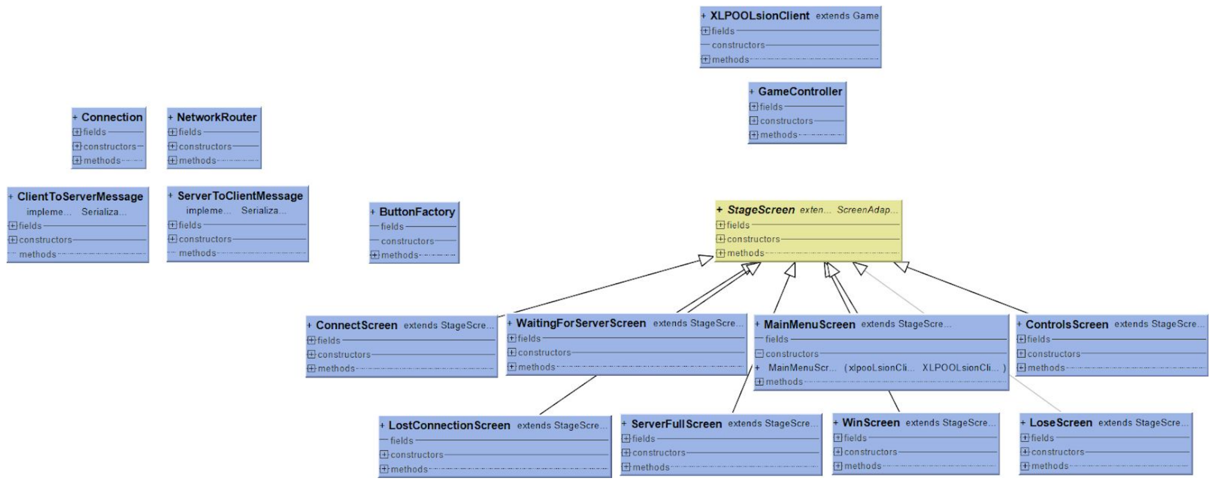
Development Documentation

António Cruz & Miguel Duarte

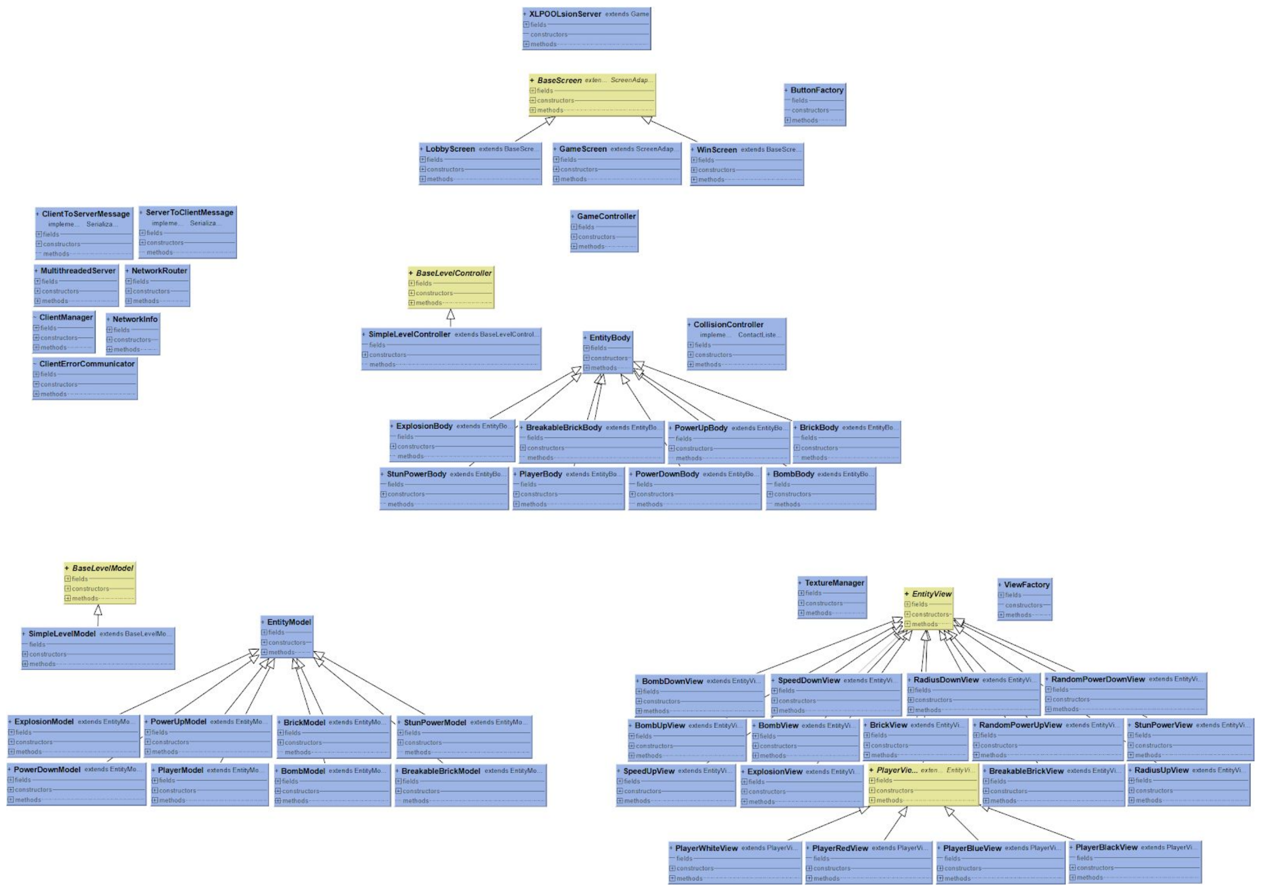
LPOO - Laboratório de Programação Orientada por Objectos
up201603526 & up201606298

Updated UML (Full UML in repository README)

Android UML


XLPOOLsionClient extends Game
GameController
Connection
NetworkRouter
ClientToServerMessage impleme... Serializa...
ServerToClientMessage impleme... Serializa...
ButtonFactory
StageScreen exten... ScreenAdap...
ConnectScreen extends StageScre...
WaitingForServerScreen extends StageScre...
MainMenuScreen extends StageScre...
MainMenuScr... (xlpoolsionCli... XLPOOLsionCli...)
ControlsScreen extends StageScre...
LostConnectionScreen extends StageScre...
ServerFullScreen extends StageScre...
WinScreen extends StageScre...
LoseScreen extends StageScre...
fields
constructors
methods

Server UML


XLPOOLsionServer extends Game
BaseScreen exten... ScreenAdap...
ButtonFactory
LobbyScreen extends BaseScre...
GameScreen extends ScreenAdap...
WinScreen extends BaseScre...
ClientToServerMessage impleme... Serializa...
ServerToClientMessage impleme... Serializa...
MultithreadedServer
NetworkRouter
ClientManager
NetworkInfo
ClientErrorCommunicator
GameController
BaseLevelController
SimpleLevelController extends BaseLevelControl...
EntityBody
CollisionController impleme... ContactListe...
ExplosionBody extends EntityBo...
BreakableBrickBody extends EntityBo...
PowerUpBody extends EntityBo...
BrickBody extends EntityBo...
StunPowerBody extends EntityBo...
PlayerBody extends EntityBo...
PowerDownBody extends EntityBo...
BombBody extends EntityBo...
BaseLevelModel
SimpleLevelModel extends BaseLevelMo...
EntityModel
ExplosionModel extends EntityMo...
PowerUpModel extends EntityMo...
BrickModel extends EntityMo...
StunPowerModel extends EntityMo...
PowerDownModel extends EntityMo...
PlayerModel extends EntityMo...
BombModel extends EntityMo...
BreakableBrickModel extends EntityMo...
TextureManager
EntityView
ViewFactory
BombDownView extends EntityVi...
SpeedDownView extends EntityVi...
RadiusDownView extends EntityVi...
RandomPowerDownView extends EntityVi...
BombUpView extends EntityVi...
BombView extends EntityVi...
BrickView extends EntityVi...
RandomPowerUpView extends EntityVi...
StunPowerView extends EntityVi...
SpeedUpView extends EntityVi...
ExplosionView extends EntityVi...
PlayerVie... exten... EntityVi...
BreakableBrickView extends EntityVi...
RadiusUpView extends EntityVi...
PlayerWhiteView extends PlayerVi...
PlayerRedView extends PlayerVi...
PlayerBlueView extends PlayerVi...
PlayerBlackView extends PlayerVi...

Design Patterns Used

These were the main design patterns used:

1. Factory
    a. Used for creating a Button Factory (for libGDX Buttons, server and client) and a View Factory (for our custom views, server)
2. Factory Method
    a. Used when creating the sprite for each view, EntityView would call the abstract method createSprite defined in its subclasses, for example
3. Singleton
    a. Used in several classes, such as the GameController class (server and client) to ensure that there was only one instance at a time of the state and event handling class, for example
4. Observer (libGDX's Box2D)
    a. The observer pattern was implemented with the use of libGDX's Box2D addon, used for physics contact listener callback functions.
5. Object Pool (libGDX container)
    a. The native libGDX container *Pool* was used in terms of creation of bomb and explosion models, due to their high rate of creation and deletion
6. Template Method
    a. Used in level creation, being that the base game logic for the level controller and model was defined in an abstract superclass. The concrete subclasses defined not different parts of logic but matters that can differ from level to level such as the block placement. These overriden methods were abstract in the superclasses and called there, thus being template methods.

Relevant Design Decisions

In order to implement the desired functionalities, some design decisions were taken, being these the most relevant:

- Serialization based communication protocol
    - This decision allowed for easier management of messages, by constructing classes that would represent them
- Every game entity should have its own View, Model and Controller (Body)

    - This decision, despite seemingly overly complicated, allowed for easier scalability in the program, due to being able to extend previous existing view/model/bodies and adding to their functionality
    - Keeping a coherent structure facilitated other matters such as collision detection by calling standardized methods in the model, generalizing code, among other benefits
- Server signals events, client communicates input
    - Despite now seeming blatantly obvious, at the beginning we were very confused about which should say what. Ensuring that only the server communicated events (game start, player lost, etc) ensured that all the clients were in sync and that there did not have to be any needlessly complex routing (client to other clients or client to server to broadcast to other clients).
    - With time it also became obvious that the client should only handle inputs, such as the player moving the joystick, pressing buttons, etc, leaving all other processing to the server

Major Difficulties along the way

These were some of the major difficulties experienced along the way:

- ➔ Lack of previous knowledge
    - ◆ Despite learning more each day (inside and outside the degree context) there were some points that we were unsure how to approach at first, due to lack of previous knowledge. This was felt a lot, for example, when designing the application architectures and the network architecture.
    - ◆ It was, however, a good reason to grow and learn.

There were also some more concrete difficulties, namely:

- ➔ How to stop explosions when the first brick was hit?
    - ◆ This was a major point in terms of game functionality and we spent some time without getting a good way to make it work.
    - ◆ Box2D was being used for physics and collision detection (which facilitated solving other issues in the game), however we found out that the collisions were not fired when the object was created on top of one another without moving, but only when one of them (at least) moved. We would also by then have no easy way to stop subsequent explosions from being created,

        which was also a problem.
    - Finally, we were reminded of a structure we had used in the first project for this course, that allowed for something similar.
    - We then decided to store all the game bricks in a separate matrix, to allow for easier verification of where the explosion should end
- How to allow for more fluidity of the controls?
    - Being able to easily control the player character is a major point of all games, and we also had some problems regarding this.
    - Eventually a good balance of all techniques tried was found, ranging from things like resizing the player body and changing its speed to changing the mobile app joystick's sensitivity and scale.

Lessons Learned

Several lessons were learned along the development of Bombin' Man, being that in this section they will be touched upon on a more general level.

1. Think first, code later
    a. This project was one of the largest scale ones we worked on and this was not on our minds since the beginning, which was a huge mistake!
    b. We very quickly found out the importance of strategizing, and started to verify our ideas with each other, even when we were working on distinct parts of the application.
2. Design patterns are useful but should not be overused
    a. After hearing in classes about how good design patterns are, we started to try and fit them in every place we could, believing that that would lead to having better code. Unfortunately, we quickly realized that this idea was wrong, and that it left our architecture looking like a Frankenstein monster!
    b. In short, nothing beats simple and clean code, not even all the good and respected design patterns.

Overall Time Spent Developing

The development of the project started early on around the mid of april during that time until the end of May we spent around 8h a week working for the project where the basic structure and design decisions were made.

During the first half of May we stopped working as hard in order to focus on other subjects, however in this week prior to the delivery both of us worked everyday on it around 6-7h a day having worked even harder in the last 3 days.

We did not keep exact track of how much time was spent but we believe that including the time involving acquiring the knowledge that was missing around 200h were spent.

Work Distribution Amongst Team Members

Work was, for the most part, distributed evenly, despite sometimes one of the group members working more, but in other times the other would work more and compensate, leaving the final balance to be 50/50.