# ÐÐ°ÑÑÐ¸Ð½ÐºÐ¸ Ð¿Ð¾ Ð·Ð°Ð¿ÑÐ¾ÑÑ Ð»Ð¾Ð³Ð¾ÑÐ¸Ð¿ rug

# *Advanced Object Oriented Programming*

# *Final Assignment*

# *Group 60*

*Nicolae Ghidirimschi & Alina Boshchenko*

*29 October 2018*

ASTEROIDS

*REPORT*

1. *Problem Description:* The task for this assignment was to use working 2D Asteroids game and extend it by the ability to play the game in multiplayer as well as adding more functionality and improvements to the game. More detailed, the task included implementing stable multiplayer functionality on the basis of UDP for the game to end when only one player is left alive. Moreover, it suggested adding a main menu in order to organize the states of the game better, a remote table showing the highest scores, and distinguishing players using nicknames and colors.
2. *Analysis:* Main menu with multiple states should be created with the opportunity to pick a specific state. Single player mode the game should run normally with the score increased every time tick. When selecting to spectate the game a user should add IP address and the port of the game that he wants to spectate if it exists. Then an instance of the game being played should be ruined on the Spectator without him altering the state of the model. Hosting a game or joining a game also require IP address and a port. When an online game ends the winner should be notified about his win, other clients return back to the Main Menu. If a player has won then his name is included in a database.
3. *Design description:* Created program contains several packages describing the content of the classes enclosed in them respectively.

* **The gui package** includes classes that deal directly with the interface of the program.

The actionListeners (hostGameActionListener, joinGameActionListener, singleGameActionListener and SpectateGameActionListener) are responsible for hosting a game, joining, running and spectating game respectively.

The MainMenu class (extending JFrame) provides Main Menu window consisting of buttons and text fields that implement actionListeners in order to launch selected game directly from the Main Menu.

* **The database package** provides a database as a storage for the highest scores. The Database class allows to implement and exploit the SQLite database that holds a list of names and scores, while class ConnectBase gets a connection for this database. SQLite pattern was chosen due to its convenient implementation, fast adding, updating and deleting data methods and opportunity to deal with multiple incomes.
* **The model** **package** includes classes responsible for changing the core state of the Game. The majority of the classes in this package were refactored or improved in a way to suit the needs of the newly created program (ex. the loggers were added).

The main concept is that every time tick the information about the game state should update and this change should be sent to all the client connected to a certain server. Server also should receive updated information from all the clients every time tick in order to understand if they are still connected or not and what actions were committed (case multiplayer). Every time we start a game a new server bound to the game model should be created via the Datagram socket that manages the UDP connections of the server.

* In order to provide spectating and joining a game opportunity we created **Client and Server packages** included into **packet package (nested in model package).**

By using a separate package **packet** that unites Client and Server we can send the whole game in one packet when asked spectating or joining.

**Client**

Any ClientGamePacket (instance of the abstract class) represents a packet which is sent by a game client and received by a server. It extends from a GamePacket, which means it inherits the SendPacket method that allows the sending of the packet from any "client" Datagram socket. Any concrete class that extends from ClientGamePacket should implement the handleClientPacket method, which takes care of the actions the server that received the packet should take.

Both ClientAskJoinPacket and ClientAskSpectatingPacket extend ClientGamePacket and contain methods describing the actions the server that received this packet should do depending on the type of what client asks.

**Server**

Any ServerGamePacket (instance of the abstract class) represents a packet which is sent by a game server and received by a client. It extends from a GamePacket, which means it inherits the SendPacket method that allows the sending of the packet from any "server" Datagram socket. Any concrete class that extends from ServerGamePacket should implement the handleServerPacket method, which takes care of the actions the client that received the packet should take.

ServerJoiningDeniedPacket and ServerSpectatingDeniedPacket (extends ServerGamePacket) contain method that describes the actions the specified client that received this packet should do in case of packet being denied (it could occur if the capacity of spectators/joiners is exceed (should be defined by the starting player in the main menu)).

ServerUpdatedGamePacket (extends ServerGamePacket) provides the actions the specified client that received this packet should do in case of the game being updated.

In this two packages the inheritance described allows to avoid code duplication and can also make application code more flexible to change if it will require more extension. With such inheritance, we are also be able to override the handleClientPacket and handleServerPacket methods of the base classes so that meaningful implementation of the base classes methods can be designed in the derived classes.

**Client and Server packages** **included in model package** implement Runnable and run as threads.

**Client** provides information about certain client and sends packets through established connection (via datagram socket)

**Server package** contains three classes. Every ConnectedClient represents a client which is currently connected to the server; it is used in the handleClientPacket methods. **Server class** allows to create a new server bound to the game through Datagram that manages the UDP connections of the server, while **MultiplayerServer** that extends Server creates a new server bound to the current Game that has already been running.

* Also some minor modifications were made to hold a color can be rendered on the screen as well as Game Over statement and opportunity to start a new game.

1. *Evaluation:* Despite the logic of the project now suits all the requirements we still need a couple of days to fix several issues regarding the networking of spectators and joiners. In addition, a little bit more attention could be payed to the improvement of the graphical parts of the game such as color selection. Also, if we had more time we would apply futures to our project and improve the database by making it remote.
2. *Team Work:* Before starting writing the code we had conversations of how to extend and modify the project and tackle all the aspects of the task given. Also we discussed how to make a good-looking interface and improve and optimize the code provided.

In order to increase productivity and save time we delegated tasks. The main concept of project and code and gui improvement was created together. Nicolae made the architecture of the networking, developed principal structure of servers and clients and improved several graphical issues. Alina dealt with the database, also contributed into networking and afterwards summarized the work done. We also want our project to be perfect so we need a little bit of time to fix all the minor issues and extend it with all the improvements we have in mind.