1. Created Display, Game and Launcher classes.

Display – initialized Jframe – the frame of our application and Canvas – the “sheet” on which we are drawing our graphics.

Launcher – initializes the Game and calls the game.start() method.

Game – contains the logic for running out game.

1. Created start and stop methods for creating new threads and joining them at application close.
2. Created init() method – for initializing stuff in out game class – in this case – initializing Display.
3. Created render() method – takes care of rendering graphics – showing images, background, animation and stuff.

Also created tick() method – takes care of calculations done before each animation/visualization.

1. Initialized BufferStrategy and Graphics within render method.
2. Added resources folder and linked it as a project library
3. Created gfx (graphics) package and created ImageLoader class for initializing/loading pictures for use in program.  
   Loaded a sample background image to try. Using .getResource(path) we convert string path to URL.Using ImageIO.read() we convert a image file to buffered image.
4. Created SpriteSheet class – cropping images from sprite sheets.
5. Created Assets class for taking images and using them as a field
6. Created logic for rendering depending on frames per second.
7. Created abstract class State – monitoring different game states – paused, running….and also 2 classes – GameState and MenuState, extending the abstract class state.
8. Created static class StateManager – manages the state in which the user is currently

Renamed Game class to GameEngine. Moved Assets.init() to the init() method of GameState class.

1. Created entities subpackage to game package.Creating Player class there.

Added static int gravity to GameState – measured in PIXELS – pulling 2 pixels downwards

The Player has a boundingBox – rectangle that surrounds our graphics and helps us with backend collisions.

Also the player has it’s own tick() and render() – he will update his values and graphics himself! It won’t be done by GameState.

Made intersects method in order to check wether player intersects with enemy

1. In GameEngine init() call for the StateManager to set the current state to a new GameState.

Since the State classes are not static, they are object and contain some values which have to be saved somewhere. That’s why we initialize them as fields of the GameEngine class in his init() method.

1. InputHandler – a class handling keyboard inputs. Implements KeyListener. Attach the listener to Canvas

Made player in GameState public static field…lector said it is better with Dependency Injection but he has no time and I have no idea what this is….

So nakovka is able to move left and right – both her animation and her boundingBox

1. Created abstract class Unit – for all unit types to inherit.
2. Created a test Enemy class and made him move around on his own – left and right, around his initial position.