Description of the project

Sections

1. **Idea**
2. **Users**
3. **Description**

Idea

A collection of the most known old arcade games: Pacman, Tetris, Pong etc. I will focus mainly on the Pacman clone during this period. If I get it quickly done I'll extend with some other game.

I'll build the games around the LWJGL (Light Weight Java Game Library) package, for OpenGL support. This will allow for higher frame rates and shading effects. Because I'm making old 2D games, I can do most of the things with simple quads, so it will not be that hard to achieve nice results.

I have also thought about the possibility for users to add their own extensions to the game by easily extending the Game class and placing their project inside a game folder in the installation directory. Then the game loader would find the instance and add it to the list of game instances.

Users

One player (for now, have no intends yet to make it multiplayer).

Description

Overall

The project is split up in three different Netbeans projects. The first project is wrapping up the LWJGL (Light Weight Java Game Library) OpenGL bindings, for easier management for the end user. The second project is the Backman game itself, one simple 2D game in a 3D space for proving the concept of the Engine project. And the third project is a central loader for all games that people might make with the Engine I provided.

Engine (LWJGL wrapper)

The Engine consists of simple abstract classes and interfaces for the user to get started quite quickly with making his/her own implementations. The game loader uses the Game class found in the Engine for instantiating the games, therefore it is critical that the user uses this class in his game. At all other aspects the user can self determine what classes to use from the engine and what to create on their own.

BackmanGame

The Backman game instantiates the Game class and creates a SceneManager to manage all the Scenes used in the game. These are for now: *MainMenuScene, HighScoreScene, GameOverScene,* *GameWonScene, GameplayScene*. If I would have had more time I would have made a *PauseMenuScene* too, for now the game only pauses with a black out faded color. Then the game class starts to update and render the currently selected scene in the scene manager. Whenever a scene is changed to another the new scene's *show* method and the old scene's *hide* method are called.

Every menu has a menu item list. For now all the menu items are only labels and buttons, but it wouldn't be that hard to make for instance sub menus and similar fancy menu stuff. The menu scene containing the menu items controls the updating and rendering of them. The menu items that are buttons have an action listener assigned to them, for smarter action handling. This way the receiver doesn't need to know of the caller. You could have done this with events too, to make the receiver know who did do the call, but it wasn't necessary in this small project.

The GameplayScene creates the necessary fields and data used in the game, such as loading assets and creating the level. This allows the menu system to load much faster when starting the game from the game hub. Even though there aren't many assets, you can clearly notice a short loading delay when clicking the new game button. Then it makes sure that everything is rendered in the correct order, to make sure nothing is rendered on top of something else (that shouldn't be covered up). This object also acts as a "traffic hub", that tells the different managers what to do when certain flags have been activated e.g. when Backman eats a super snack, this is noticed by the SnackManager and the GhostManager needs to know about this, to be able to kill the ghosts when Backman collides with them.

The game is won when all the yellow snacks are eaten. You get bonus points from eating fruits and killing ghosts. When you eat cherries the camera rotates around the Y-axis, to make the X-controls inverted, to get the world to rotate back you'll have to eat an apple. These actions are just extended snacks, with an custom texture and an assigned special action listener. You get minus points from being killed. If you are killed three times you are prompted with the game over screen. Only high scores from won games are stored in the high score list.