GET OFF MY LAWN!

AUTHORS

Nicolai Skovvart (nbsk), Benjamin Ma (benm), Peter Ølsted (ptoe)
Project: https://bitbucket.org/PeterOelsted/get-off-my-lawn

ABSTRACT

A tower defense game inspired by Popcap's game Plants vs Zombies.
It is a lane game where you have to balance your economy and your positioning and power of your defensive structures to protect your lawn against the invading kids. To be successful you have to master your build-order, build in the right positions and utilize abilities at the right time to complete the levels.

REQUIREMENTS

The complete set of your mandatory (must) and secondary (may) requirements.

MANDATORY REQUIREMENTS

Accurately simulate the concept of Plants vs Zombies
Have an interactive graphical user interface
Use audio as in the original game
Persistently store level progress, unlocked towers and discovered attackers
Report on all players stored data
Towers and attackers must be animated

Several different attackers and towers, each with unique behaviors
Enable players to create profiles having individual level progress and unlocks
Must be awesome

SECONDARY REQUIREMENTS

Multiple levels with an increasing amount of different attackers and towers. Each completed level should add a new element to the game, such as a new defensive structure or a new attacker, or something similar.
Achievements/Highscore

MAYS

A copy of the character Crazy Dave from Plants vs Zombies
Bonus minigame(s)
Installer

OVERVIEW

We intend to do a tower defense game inspired by Plants vs Zombies. In our implementation instead of plants fighting zombies, it is an old man defending his lawn from kids with various junk he’s gathered through the years. An example of this could be a fishing rod..

The basic game revolves around building the proper defense against the attackers while making sure your economy will grow at a rate to accommodate building the required defense. Using abilities may help when you’re in a pinch. All of these things will progress as you advance through the levels, that get increasingly difficult.

HIGH LEVEL VIEW

The game is written in C# using XNA as a graphical framework and XML for storing data about the levels, kids, towers, media-resources and game-settings. Player information will be stored in a harder-to-manipulate format.

The system is based around a synchronous message parsing architecture. The update and rendering of every gameobject (visible or not) is done using a component system (we call Behaviors). Each game object is simply a collection of behaviors and attributes. Each behavior can subscribe to one or more of the different passes the game uses. Examples of these passes could be passes where animations are updated, physics are updated or objects are rendered. If a gameobject was to interact with another gameobject, it has to send a message with a stringkey identifying the type of message, and the variables the target gameobject needs. Passes are used for consistently updating objects during the update loop.

ROLES

Architecture: Peter
Concept design: Nicolai

Blackbox testing: Benjamin
QA: Nicolai
XML-LINQ-Wizard/Media-loading: Nicolai
Coding: All of us
Art and sound: Benjamin

DICTIONARY

GOML: A shorthand for Get Off My Lawn, the name of our application.

Behavior: Behaviors have a parent that they can communicate with, and can react when messages are passed to them and when the game-loop is updated.

GameObject: GameObjects have a unique ID (assigned by a GameObjectManager), a list of behaviors and a collection of attributes that are stored with a stringkey. GameObject can receive messages and forward the message to all of its behaviors.

Attribute: Attributes are used as dynamic fields. They have the advantage of not requiring objects to move pointers to new values when the value of the Attribute is updated.

StringKey: StringKeys are a simple wrapper type for faster comparison of strings, based on the hashcode of the StringKey instead of the char-sequence of strings. They can also be used in a similar fashion as static constants.
-Types: The types messages can have.
-Parameter: Parameters are used to identify message and gameobject attributes.
-XnaType: Used to load the AssemblyQualifiedName for Xna types such as Song, Texture2D and SpriteFont. Dynamically loading types outside the assembly and mscorlib requires the AssemblyQualifiedName (example: typeof(Song).AssemblyQualifiedName) which is both long, unreadable and prone to getting changed, so they are stored in a stringkey so they can be used for reflection-based method-invocation.

Pass: Passes are used to ensure when and how behaviors are used. Some behaviors might only want to react to some of the passes, such as animation-specific passes, and some behaviors might want some processing to be done in one pass and some other processing done in another.

ObjectType:GameObjects can only have a few types. Currently they are attacker (for kids), defender (for towers) and neutral (for everything else).

Lawn: A lawn is a 2-dimensionel array of positions based on a column and a lane where you can use towers and abilities.

Hitbox: A hitbox is a rectangle bound to a position that is used for detecting collision. For example, the collision of a projectile-hitbox with an attacker-hitbox.

EXAMPLE

Run the game executable. If this is your first time running, you'll be asked to create a user which your milestones in the game will be linked to. Otherwise you'll be logged into the last created user (with the option of switching to other users or creating a new user).

There will be a couple of buttons, mainly continue campaign, view high-score, options, exit game and create new player. Picking continue campaign will continue from the last level the current user played. Starting a level will show the types of opponents you will be fighting, and you will be able to select what defensive structures you want to use.

Starting the level will make attackers start spawning, and you will have to build defensive structures that will both boost your economy so you can build defensive structures faster, as well as defensive structures that can repel the attackers.

As the level progresses more and harder attackers will spawn more frequently. At the end of the level you will receive a reward, like a new defensive structure you can build, a new ability you can use, another slot for a defensive structure or other fun or useful things.

BIBLIOGRAPHY

Citations to any references you used during the project.

Game Engine Architecture, 2009 - Jason Gregory

REVISION HISTORY

15-12-2010: Project proposal finalized.

12-11-2010: Submission for improvement

PROBLEMS DURING DEVELOPMENT

VERSIONING CONTROL SYSTEM

We've used Mercurial, visualHG, tortoiseHG for version control, and we've run in to some problems with them. A good chunk of the code has been merged manually using dropbox to transfer files.

TESTING

PEX

We used PEX to test the entire project, but after we closed the project from mercurial they vanished and all tests were lost.

Without proof we can state that the only errors found by PEX were missing checks for null values, which hasn't been an issue while running the program.

ADDITIONAL NOTES

Despite our issues with PEX, we have played the game extensively without spotting any unintentional behavior and we feel confident that there are few to no serious programming problems (causing crashes at least). We believe the biggest flaws of the current game is balancing between towers and kids, as well as the lack of limits on towers (such as a cooldown, what towers you can build per level, etc).