Sample Video Game & Sound

Beat the Bugs!
The Plan

1. Game Theme
2. Game Structure
3. Sprites
4. Trackers
5. Collisions
6. Score
7. Levels
8. Splash Screens
9. Design
10. Implementation
11. Testing/debugging
12. Documentation
13. Enhancements
14. Release
Game Theme

• Who is your audience?
• Why would they want to play your game?

A catching theme for a simple game can make the game better than a more complicated game with no theme.

Think: 'Where the hell is that Cat Shit?!?!'
Game Structure

- One player or two? Turn taking or simultaneous? Can the other player be the computer?
- Symmetric or asymmetric opponents?
- Action or Strategy?
-
Sprites

- What ImageSprites do you want?
- What StringSprites would be useful?
- Do you want to make custom Sprites?
- Which Sprites may need motion blur?

For choosing ImageSprites, be aware that they behave as squares with images painted on. This is particularly important for collision detection.

More exact boundaries are made when customizing Sprites, which can also be done with ImageSprite to get tighter boundaries.
Trackers

• How are the Sprites going to move?
• Are single Sprites going to use multiple Trackers in sequence? If so, consider using the Alarm, based either upon time or another event.
• What information do the Trackers need?
  – other Sprite locations?
  – boundary information?
Collisions

Collision of objects is the most frequent event in a game.

• What happens when objects collide?
  – Do they bounce?
  – Does one or both objects disappear?
  – Does the score change?

• How important is the resolution of the collision?
  – Are bounding boxes sufficient? If so, they are good because they are fast and simple.
  – Is normal information required at time of impact?
Score

• How can progress in the game be quantified?
• Does score trigger events? If so, use Alarms.
• People love scores. They can compare them to friends scores, play to beat the best score, or just use them to see that they are getting better.
Levels

• Levels are like integral scores.
• Many options for varying the level
  – change of theme/sounds
  – change of difficulty
  – change of structure
• Levels make the game more interesting, and more interesting levels drives players to play games repeatedly to reach those levels.
Splash Screens

• Provide smooth transitions between game levels, starting and ending the game.
• Can include more complex movement and sound because they are scripted and not interactive
• Can be reused with slight modifications
Design

• What classes are needed?
• What instance variables are needed in the classes?
• What are the methods of the classes?
• How do the classes fit together?
• Is this design flexible?
  – Can you get the basic structure completed quickly?
  – Can the basic structure be incrementally implemented?
  – Can the basic structure be enhanced?
Implementation

• If the design was done well, this should be the simple part.

• Should be done in small, separate, testable increments.

• Try to always keep a functional game ready for release – remember, programs are released, not finished!

• Will usually take longer than expected.
Testing/Debugging

• Your game almost never works the first time
• Half or more of your time may be spent here, depending on your planning and design.
• Debugging/Testing can be unpredictable in how much you'll need to do and how long it will take.
• Again, this usually takes longer than expected.
Documentation

• Document minimally as you code so as not to forget what you've done.
• Wait until the code is near completion for full documentation as you may end up documenting code that is later changed or not used.
• Develop a user manual and programmer documentation using Javadoc and web pages.
Enhancements

• If you have extra time at the end, you can make a big difference.

• It's not unusual to complete 90% of the functionality in the last 10% of the coding.

• Slight modifications in the look and function of the interface can make the game much more appealing with little effort.

• While it's tempting to continue adding functionality, you should really stop early to polish your work.
Release

- Tell all your friends about your game.
- Visit gaming sites and link up your game to their indices.
- Be prepared to get emails about your work.
- Provide source code if possible to help others code.
Beat the Bugs!

1. Game Theme
2. Game Structure
3. Sprites
4. Trackers
5. Collisions
6. Score
7. Levels
8. Splash Screens
9. Design
10. Implementation
11. Testing/debugging
12. Documentation
13. Enhancements
14. Release
Game Theme

For my game, I wanted something students would enjoy playing. I also wanted something the pertained to what I was teaching.

Students always write bugs in their programs and always get frustrated at some point. I decided to make a game they could play to relieve some frustration and have some fun.
Game Structure

Since I'm always short on time, I decided to choose something similar to what I already had coded, a shoot'em up.

It would be one player action game with the player at the bottom and enemies at the top. The player would be able to shoot the enemies and the enemies would be able to shoot at the player.
Sprites

• For the player, an ImageSprite of myself
  – I used a program to crop out the background and make a transparent gif

• For the player bullet, an ImageSprite of my head
  – I took the same photo and cropped out just my head into a transparent gif.

• For the enemies, a 'Java' StringSprite

• For the enemy bullets, a 'bug' StringSprite
Trackers

• I wanted the enemy Sprites to move in a pattern, but not in unison. I also didn't want their pattern to be monotonous.

• I chose a circling pattern for their basic movement.

• To avoid monotony, they would spin out and back periodically.

• The bullets would travel in a line and spin via the ProjectileTracker.
Collisions

• When the enemy bullets hit the player, reduce the life by one and the bullets disappear.

• When the player bullet hits the enemy, both the player and the bullet disappear.

• For simplicity, the each enemy and the player have only one bullet.
To be continued...

Next lecture