Project – Games Development

Jim Holmström
Project Description

- Goals
  - Get a basic understanding of the game development process.
  - Extend/Create a 2D platform game.
  - Work as a group, dividing the work.
Project Group

- **Roles**
  - **Project Manager** (organize the group)
    - Organize the group
    - Meetings
    - Responsible for maintaining time reports
      - Members should submit time reports to PM
  - **Other roles** – not necessarily one-to-one
    - Level designer
    - Graphics
    - Game play / story
    - Coder
Project Group

■ Group friction?
- Remember, it is a team effort.
  - “I did my part so I’ll just sit by and watch”
- If you have any problems with the group, contact us immediately and we will try to solve them.

■ Other problems?
- Contact Jim or Justin,
  gamescoursestaff@list.it.uu.se
Planning is important

- Before coding…
  - What game do you want to make
    - Strategy, build and destroy, adventure, action/shootem-up, etc.
  - List features of the game
    - Multi-play or Single play
    - AI
    - Movements and Interaction
    - Physics
    - Sound/music, etc…
  - Milestones and time plan
First Meeting – Action plan

- Week 26    (Thursday and Friday)
  - PM makes an appointment with Jim.
  - The entire group meets and we discuss your *action plan*, that the group has prepared.
  - The Action plan contains,
    - Description of the game idea
    - Milestones and time schedule
    - Story-board / feature list
    - Work division, who is responsible for what
Final Meeting – Examination

- W33 (August)
  - PM book the meeting
  - Demo your game
  - Answer questions about implementation
  - All members must attend
  - Account for work division and time spent
Project Tools

- CVS – code version management
  - TortoiseCVS (windows)
    - Graphical interface integrated into Explorer
  - Unix/cygwin
    - Command line cvs

- Graphics
  - Inkscape – vector graphics
  - Paint.NET – simple paint program

- ANT
  - Build tool
**CVS - Concurrent Versions System**

- Handles multiple users working on the same file(s)
- Merges files
- Allows getting different versions
- User typically get latest versions make changes and commits them, making them available to other users.

- Important to commit only working code

![Diagram showing CVS workflow](image)
CVS

- Remote and shared repository
- Most used commands
  - `cvs <command>`
  - First time get files by `cvs checkout game`
  - To get latest changes use `cvs update`
  - Submit changes to files by `cvs commit`
  - Add new files `cvs add *.java` then commit to make it happen.
- Must have a correct CVSROOT
  - `:pserver:spel06_NN:pass@cvs.srv.it.uu.se:/spel06_NN`
    or
    `:pserver:spel06_NN@cvs.srv.it.uu.se:/spel06_NN`
  - Module to checkout: `game`
  - Passwords distributed to each group.
TortoiseCVS

- Right click to access menu
TortoiseCVS

- Adding files, committing changes

A new file was created, add it to the repository.

doc.txt was modified, commit to the repository.

Note the difference in icons.
TortoiseCVS

- Get updated files from the repository
  - Changes will be automatically merged
  - Conflicts are handled manually.
CVS – Resolve Conflict

• User 2 – cvs update
• User 1 – cvs update
• User 1 – Changes row 2 of hej.txt to:
  Tjenis
• User 1 – cvs commit
• User 2 – At the same time, changes row 2 of hej.txt to:
  hoppla
• User 2 – cvs update (commit will give error due to not up to date)
• User 2 – resolves conflict

hej.txt:  
hejsan
<<<<<<<<< hej.txt
hoppla
========
tjenis
>>>>>>>> 1.4

choose

hej.txt:  
hejsan
hoppla

hej.txt:

hejsan

• User 2 – cvs commit
Tools

- Other tools
  - IDE - Integrated Development Environment
  - Inkscape – vector graphics drawing program
  - Paint.NET – image editor
  - ANT
    - Some IDEs have automatic generation of ant build scripts.
    - Can create jar files for you, very convenient
ANT – simple example

Assume we have the following code

In ./src/mypkg/HelloWorld.java

```java
package mypkg;

public class HelloWorld
{
    public void main(String[] a)
    {
        System.out.println("Hello World!");
    }
}
```
ANT – simple example

./build.xml:

```xml
<project>
  <target name="clean">
    <delete dir="build"/>
  </target>

  <target name="compile">
    <mkdir dir="build/classes"/>
    <javac srcdir="src" destdir="build/classes"/>
  </target>

  <target name="jar">
    <mkdir dir="build/jar"/>
    <jar destfile="build/jar/HelloWorld.jar" basedir="build/classes">
      <manifest>
        <attribute name="Main-Class" value="mypkg.HelloWorld"/>
      </manifest>
    </jar>
  </target>

  <target name="run">
    <java jar="build/jar/HelloWorld.jar" fork="true"/>
  </target>

  <target name="all" depends="compile, jar" />
</project>
```
ANT – simple example

- Using ant
  - ant compile
    - Compiles Helloworld.java and place the class-file in ./build/classes/mypkg
  - ant clean
    - Removes the build directory
  - ant jar
    - Creates a jar file with mail class Helloworld.jar
      - Can be run with java -jar Helloworld.jar
  - ant all
    - Dependencies on compile and jar
  - ant run
    - Run the created jar file
Groups!