Game Development

Pr. Olivier Gruber

olivier.gruber@univ-grenoble-alpes.fr

Laboratoire d'Informatique de Grenoble

Université de Grenoble-Alpes

Overview

- Your game – Our software requirements
  - Everything you need to know...
- Graphical User Interface (GUI) principles
  - Screen, pixels, and widgets
  - Event-oriented programming
- Game specifics
  - A game is the simulation of a virtual world
  - With 2D or 3D rendering of the virtual world
  - Managing user interations
  - With automata for “artifical intelligence”

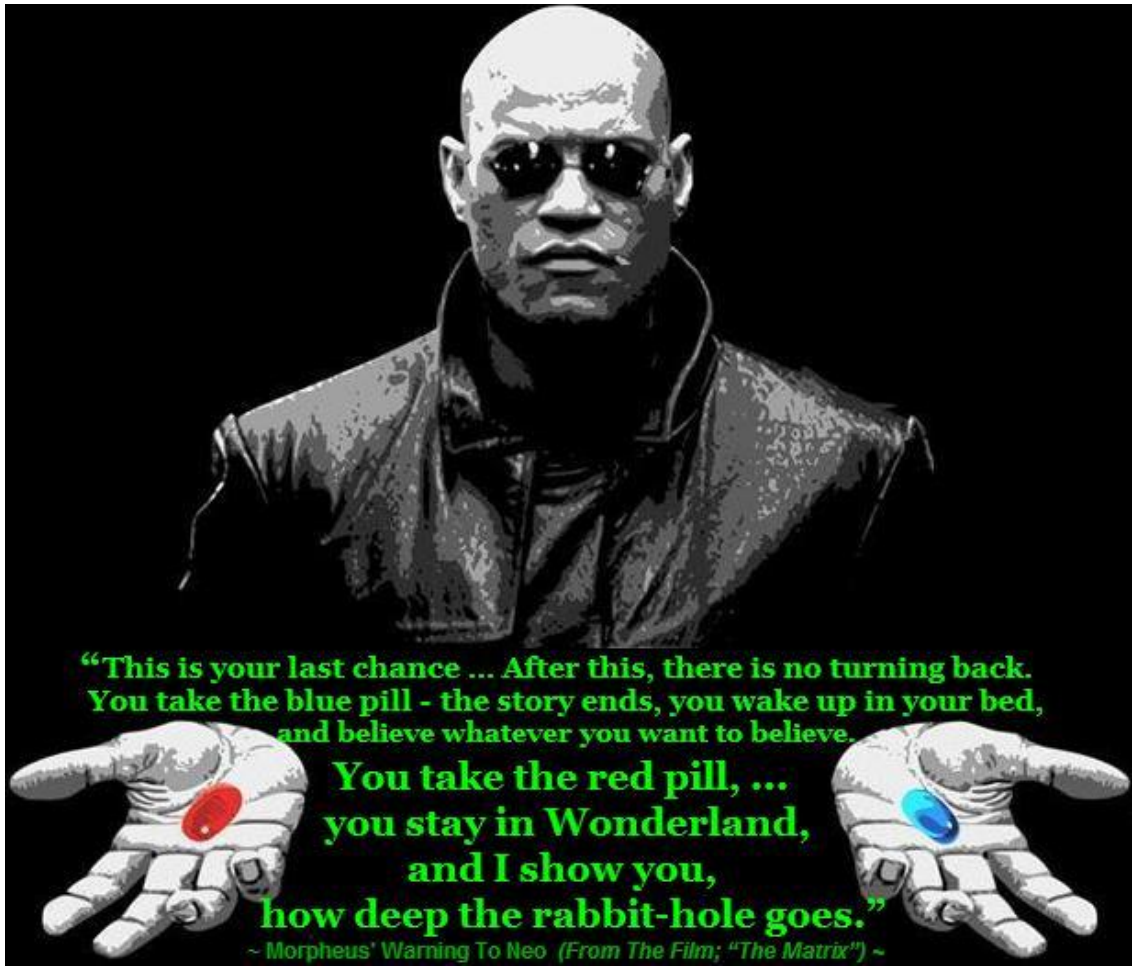
Your game – Our software requirements

- Your game is your game...
  - From a gaming perspective, its style, etc.
- Our software requirements
  - You will code in Java
  - You will use the game framework that we will provide you
  - The framework is essentially based on Swing/AWT
- Software goals
  - Learn to work as a team
  - Improve your Java development skills
  - Learn the basics of graphics
  - Master automata programming

Your game – Our software requirements

- You will use a repo git on github
  - You will work as a team, so expect it to be a **plus** and a **minus**
  - Regular commits (every day or so)
- The group lead will maintain a worklog in the repository
  - A ToDo list, who is doing what, etc.
- You will give me full access to your repo
  - I will be working with you, helping and evaluating
  - **We will discuss your code, so be ready to explain your code in details**
  - You are expected to know "*enough*" about the code of others
  - **You must understand the given framework, in details**
- I will help, but I will keep it fair
  - I will not code for you and not debug your code for you
  - But I will help you learn and understand

Remember, you took the red pill…



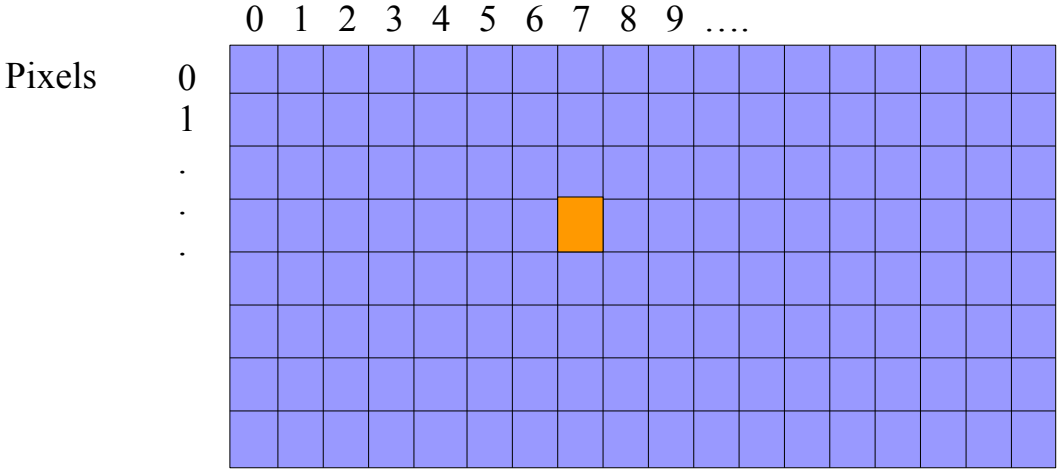
You have to convince me that you are a decent software developer

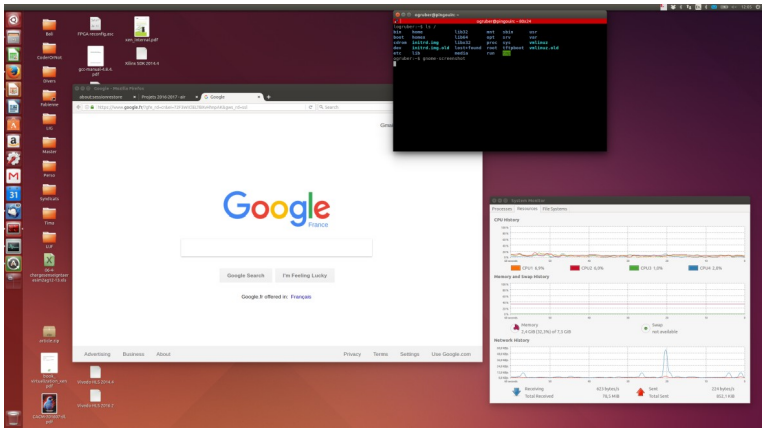
Overview

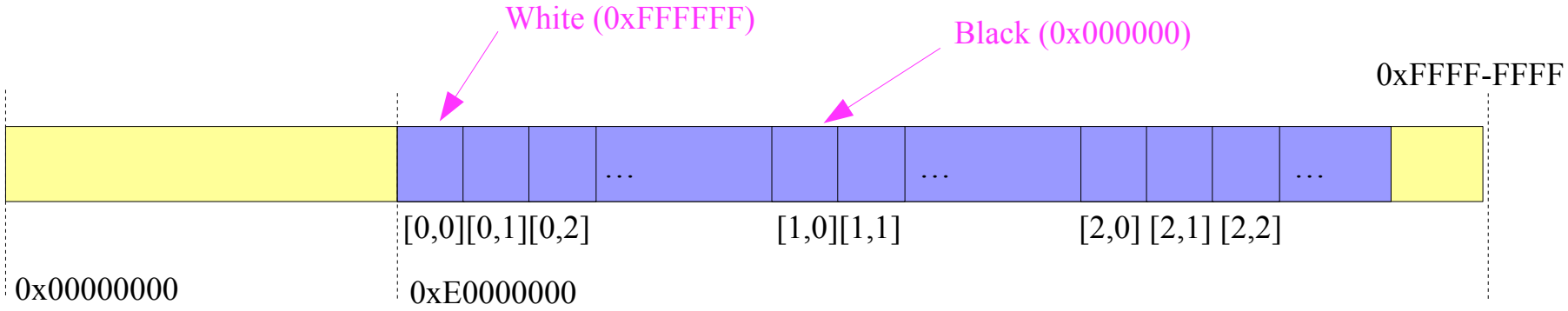
- Your game – Our software requirements 
  - Everything you need to know...
- Graphical User Interface (GUI) principles 
  - Screen, pixels, and widgets
  - Event-oriented programming
- Game specifics
  - A game is the simulation of a virtual world
  - With 2D or 3D rendering of the virtual world
  - Managing user interations
  - With potential needs in artificial intelligence

Video Buffer

- Pixel matrix
  - Each pixel encoded on 32 bits = Red-Green-Blue-Alpha
  - Video buffer in memory

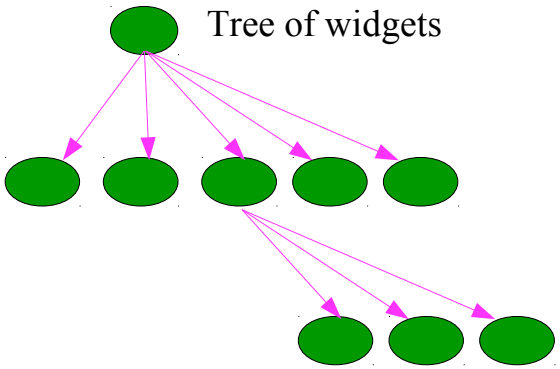


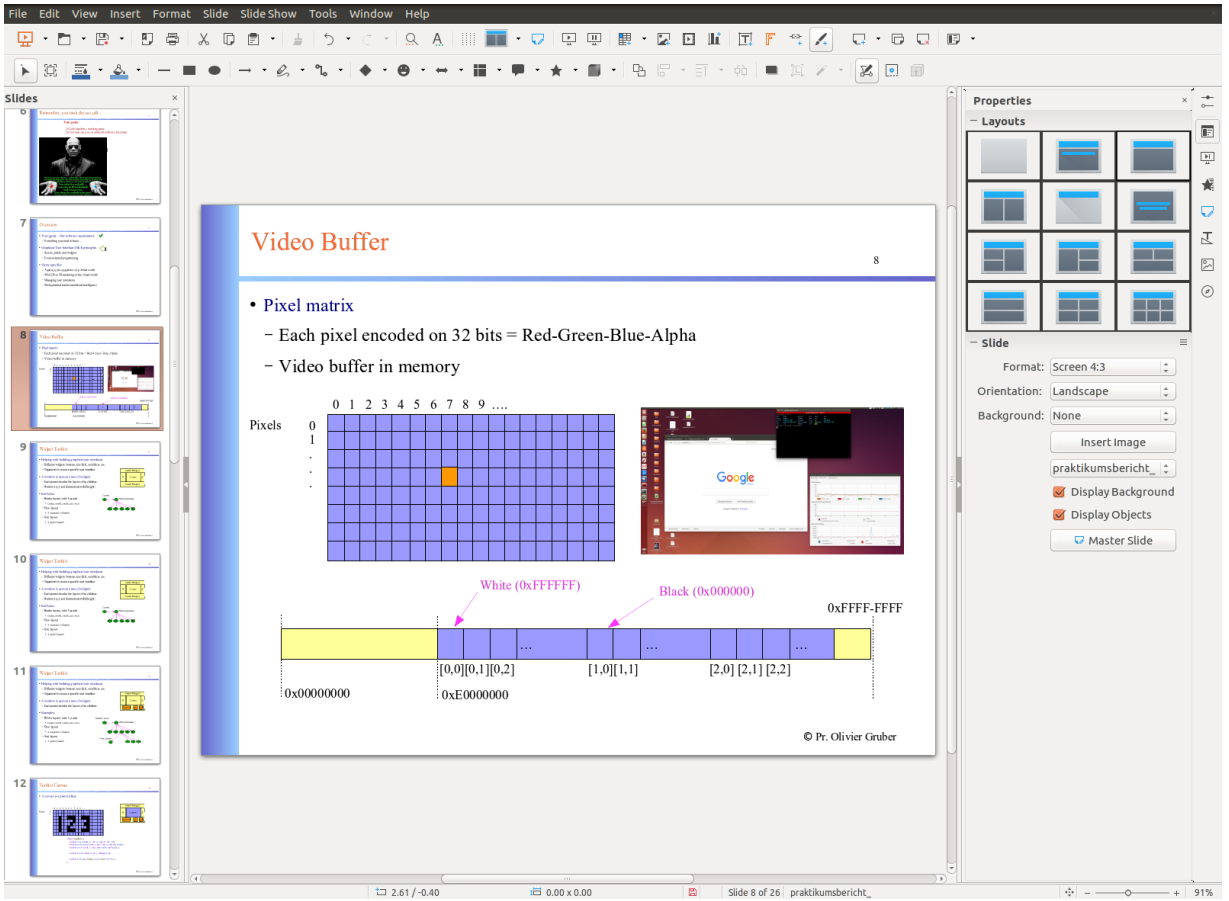




Application Window

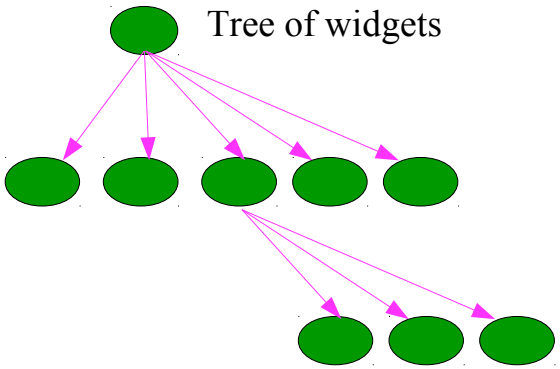
- A tree of widgets
  - Containers, buttons, text fields, images, etc.

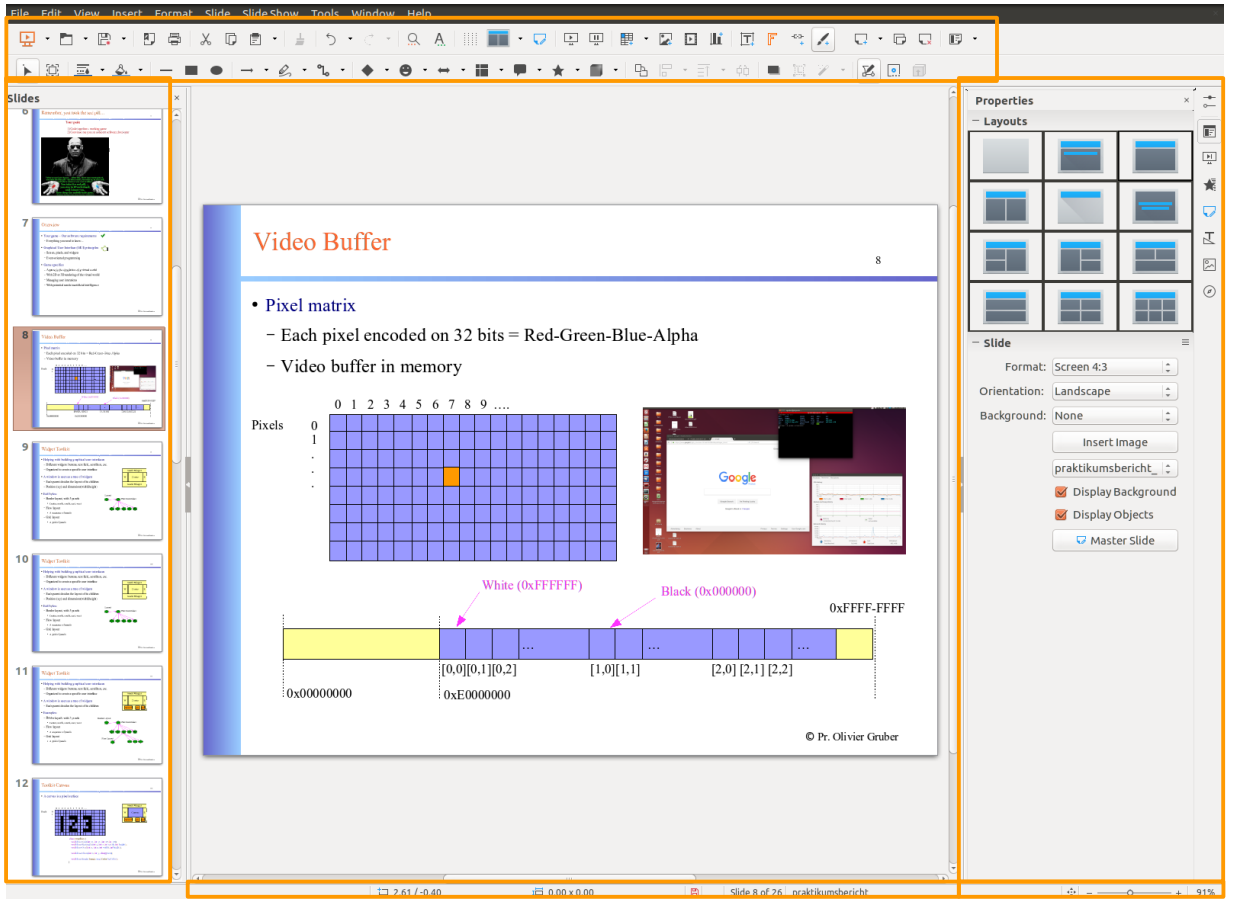




Application Window

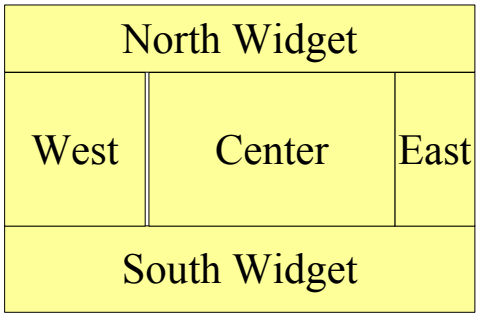
- A tree of widgets
  - Containers, buttons, text fields, images, etc.

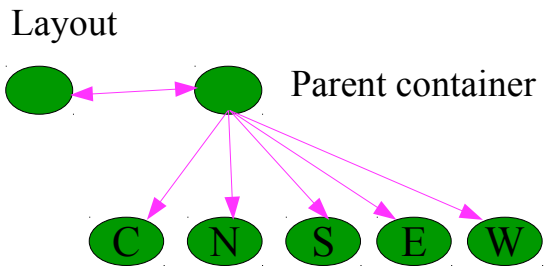




Widget Toolkit

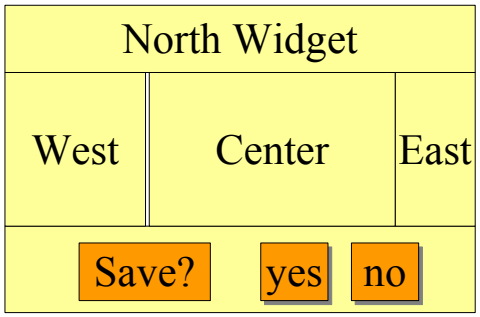
- A window is seen as a tree of widgets
  - Each parent decides the layout of its children
  - Position (x,y) and size (width,height)
  - Each children has preferred size
- Class hierarchy
  - Core classes: Container, Component, Layout
  - Widget classes: Button, Label, TextField, etc.
- Layout examples:
  - Border layout, with 5 components
  - Center, north, south, east, west

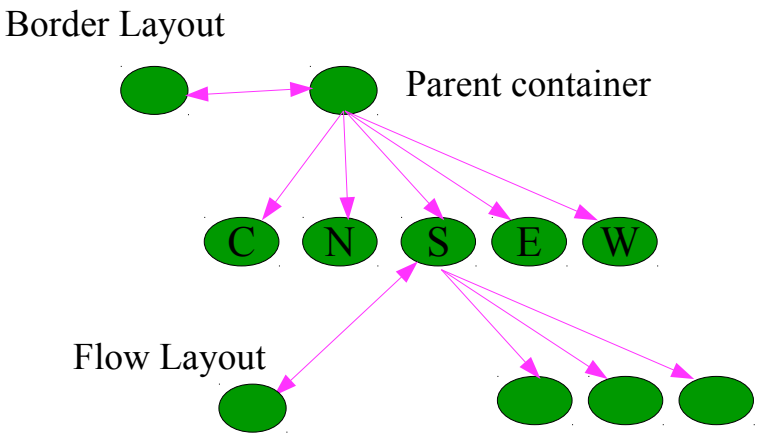




Widget Toolkit

- Different layouts
  - Border layout, with 5 components
    - Center, north, south, east, west
  - Flow layout
    - A sequence of components
  - Grid layout
    - A grid of components



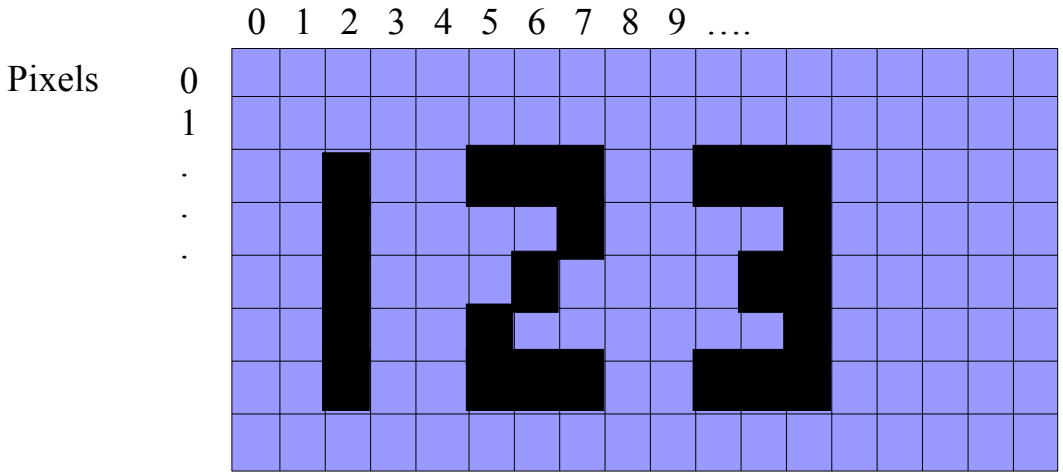


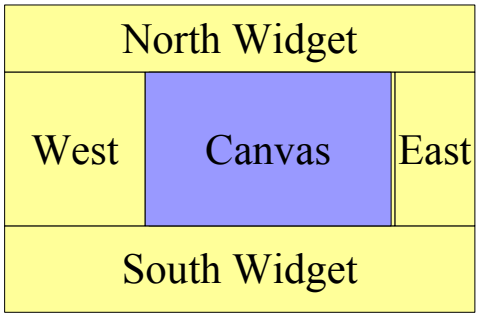

```
JFrame f;
Button b1,b2;
void createWindow() {
    JFrame f = new Jframe();
    f.setSize(800,600);
    f.setLayout(new BorderLayout());
    Container c = new Container();
    f.add(b,BorderLayout.SOUTH);

    c.setLayout(new FlowLayout());
    b1 = new Button();
    c.add(b1,BorderLayout.SOUTH);
    b2 = new Button();
    c.add(b2,BorderLayout.SOUTH);
}
```

Toolkit Canvas

- A canvas is a pixel surface






```
class Graphics {
   void drawLine(int x1, int y1, int x2, int y2);
   void drawRectangle(int x, int y, int width, int height);
   void drawOval(int x, int y, int width, int height);

   void drawChars(int x, int y, char[] text);

   void drawImage(Image img, Color bgColor);
}
```

```
class MyCanvas extends Canvas {
   void paint(Graphics g) {
      ...
   }
}
```

Programming Model

- The Hollywood principle – *‘‘don’t call us, we will call you‘‘*
  - **You do not control the flow of execution – *‘‘they‘‘* do**
  - Whose *‘‘they‘‘?* The widget toolkit
- Event-oriented programming
  - Create a window
  - Register listeners
  - That’s it…
- **Toolkits are not thread safe**
  - **Do not create your own threads**

```
class Game {
  void main(String args[]) {
    createWindow();
    return;
  }
  void createWindow() {
    JFrame f = new Jframe();
    f.setSize(800,600);
    f.setLayout(new BorderLayout());

    Button b = new Button();
    f.add(b,BorderLayout.SOUTH);

    ActionListener l;
    l = new MyButtonListener(b);
    b.addListener(l);
    return;
  }
}
```

Programming Model

- Different events
  - Keyboard events:
    - key pressed or released
  - Mouse events:
    - Enter, leave, move, button pressed or released
  - Timer events:
    - Called periodically, like every 1ms
- Event listeners
  - An event occurs
  - Your listener reacts and then returns
  - A listener is an object
  - **A listener implements a finite state automaton**

```
class MyButtonListener extends ActionListener {
  Button m_b;
  MyButtonListener(Button b) {
    m_b = b;
  }
  public void actionPerformed(ActionEvent e) {
    ...
    return;
  }
}
```

Programming Model

- Events versus threads
  - Threads run an algorithm, this is what you are used to
  - Events will feel strange at first
  - The kitchen metaphor can help you understand the difference

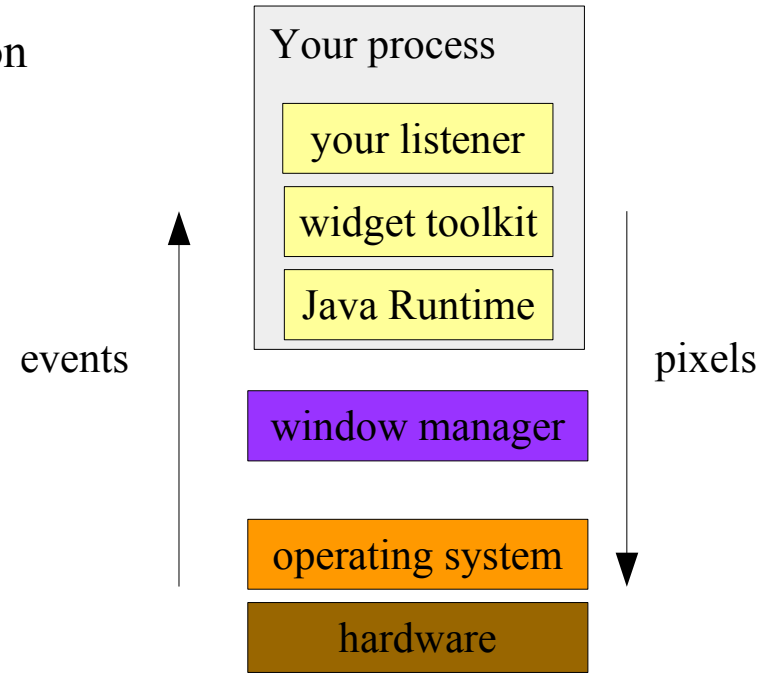
A cook follows one recipe...

More than one recipe?
Need more than one cook.

```
class MyAlgorithm {
  static void main(String args[]) {
    for(… ; … ; … ) {
      for(… ; … ; … ) {
        for(… ; … ; … ) {
          ...
        }
      }
    }
    return;
  }
}
```

Programming Model

- Events versus threads
  - An event occurs, added to the event queue of a scheduler
  - The scheduler pops the first event and executes its reaction
  - The reaction runs to completion
  - The reaction may create new events
- External events
  - Timer events are a classic…
  - Graphical toolkit events…
  - These originate from hardware events (interrupts)

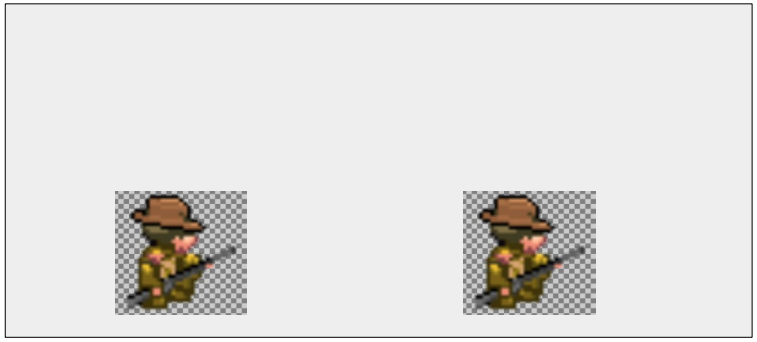


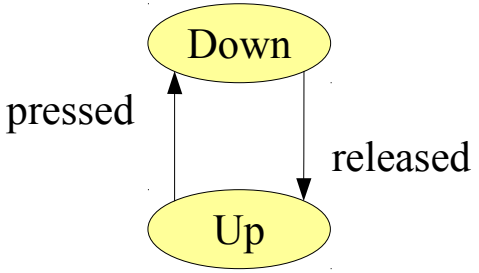
A cook reacts to events
Works through different recipes one step at a time

May need to take notes…
- what was that timer for already?

Programming Model

- Example: select a cowboy
  - Events: mouse events → (x,y) and button states
  - (x,y) → cowboy



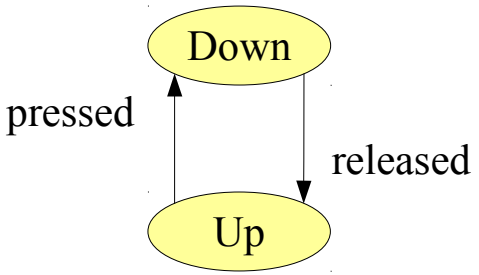



```
class Cowboy extends Entity {
  int state; // 0 is up, 1 is down
  void pressed() {
     state = 1;
     return;
  }
  void released() {
     state = 0;
     return;
  }
}
```

NEVER WAIT !
NEVER SLEEP !

Programming Model

- Example: select a cowboy
  - How do we manage a toggle selection?

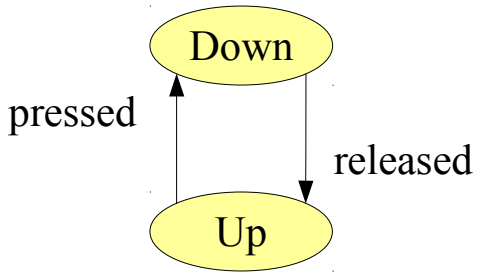



```
class Cowboy extends Entity {
  int state; // 0 is up, 1 is down
  boolean selected;
  boolean selected() {
    return selected;
  }
  void pressed() {
    state = 1;
    while (state==1)
      ;
    selected = true;
    return;
  }
  void released() {
    state = 0;
    return;
  }
}
```

NEVER WAIT !
NEVER SLEEP !

Programming Model

- Example: select a cowboy
  - How do we manage a toggle selection?




```
class Cowboy extends Entity {
  int state; // 0 is up, 1 is down

  boolean selected() {
    return state == 1;
  }
  void pressed(Event evt) {
    state = 1;
    return;
  }
  void released(Event evt) {
    state = 0;
    return;
  }
}
```

Does this work?

What about the mouse leaving and entering your button?

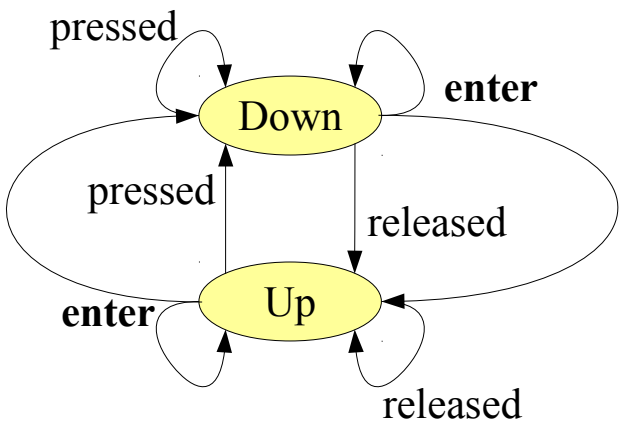
Is the automaton complete?

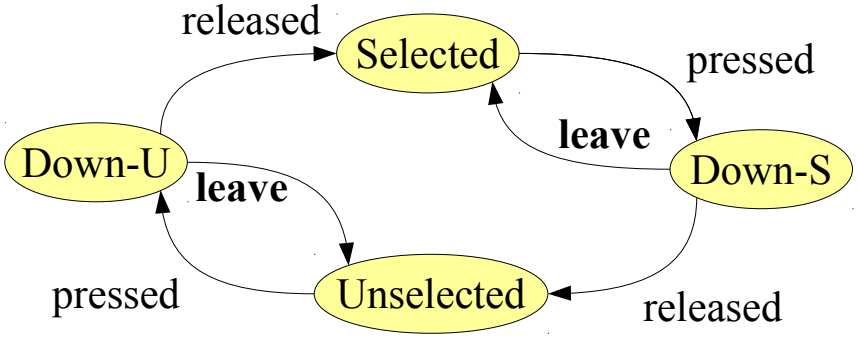
Programming Model

- Example: select a cowboy

```
class Cowboy extends Entity {
  int state; // 0 is up, 1 is down
  boolean selected;

  void pressed(Event evt) {
     ...
  }
  void released(Event evt) {
     ...
  }
  void enter(Event evt) {
     ...
  }
  void leave(Event evt) {
     ...
  }
}
```



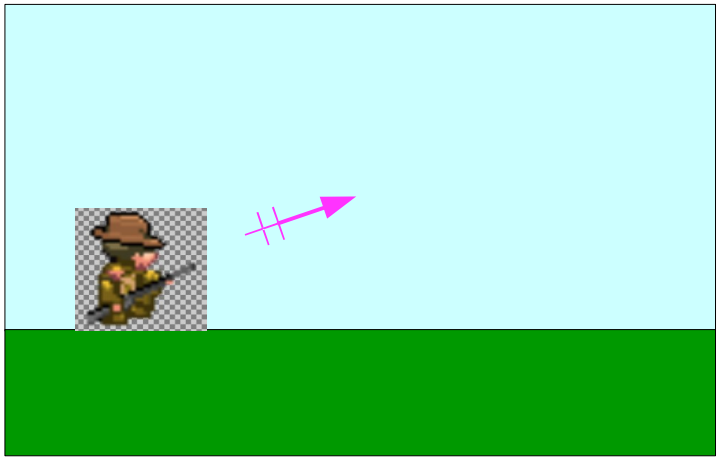


Programming Model

- Event-oriented programming and passing time
  - Flying an arrow… or a bird...
  - As time passes, the position and direction changes
  - Your code must react to the tick event

```
class Arrow extends Entity {
   float fx,fy,fz; // speed vector
   float x,y,z;     // position
   long last;       // last tick

   void tick(long now) {
      long elapsed= now – last;
      // update the position
      …
      // update the speed (gravity effect)
      …
      return;
   }
}
```

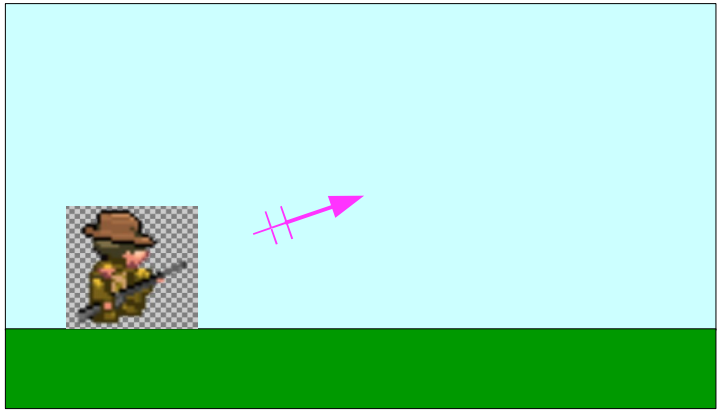


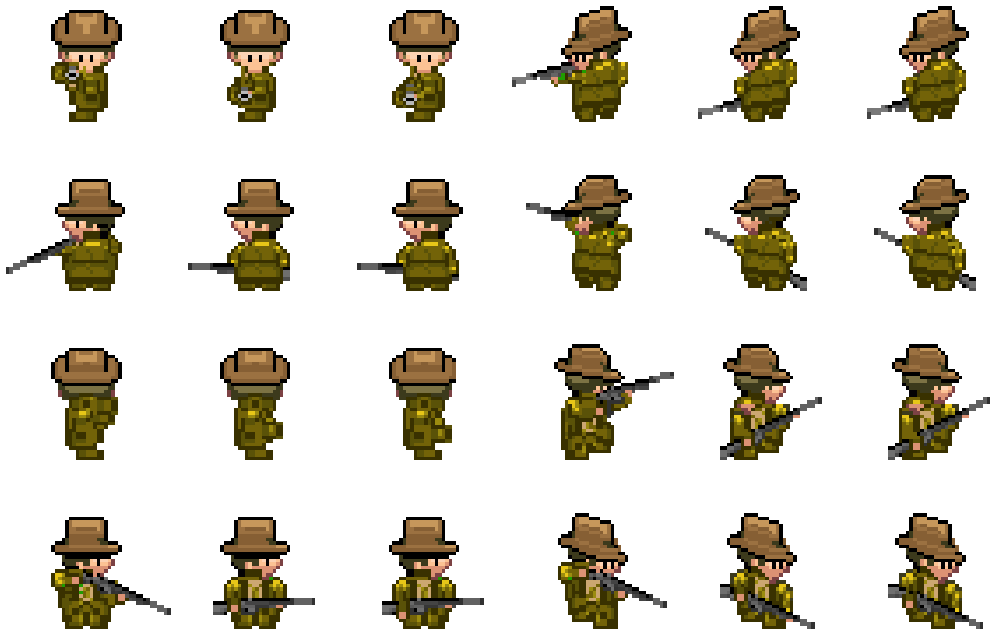
Programming Model

- Event-oriented programming and passing time
  - Animations...

```
class Cowboy extends Entity {
   float fx,fy,fz;       // speed vector
   float x,y,z;          // position
   Image images[];  // all images
   Image current;

   void tick(long now) {
      long elapsed= now – last;
      // update the position
      …
      // update the current image
      …
      return;
   }
   Image  getImage() {
      return current;
   }
}
```





Overview

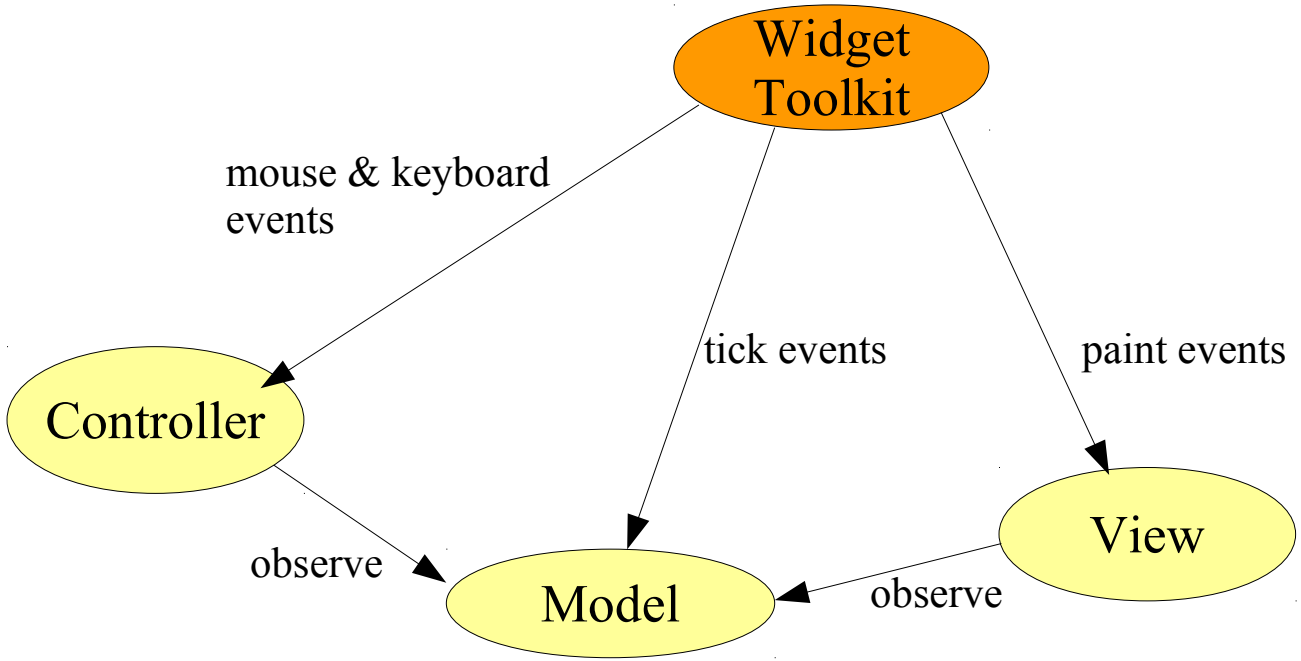
- Your game – Our software requirements ☑
  - Everything you need to know...
- Graphical User Interface (GUI) principles ☑
  - Screen, pixels, and widgets
  - Event-oriented programming
- Game specifics 
  - A game is the simulation of a virtual world
  - With 2D or 3D rendering of the virtual world
  - Managing user interations
  - With potential needs in artificial intelligence

A Game --- Software Architecture

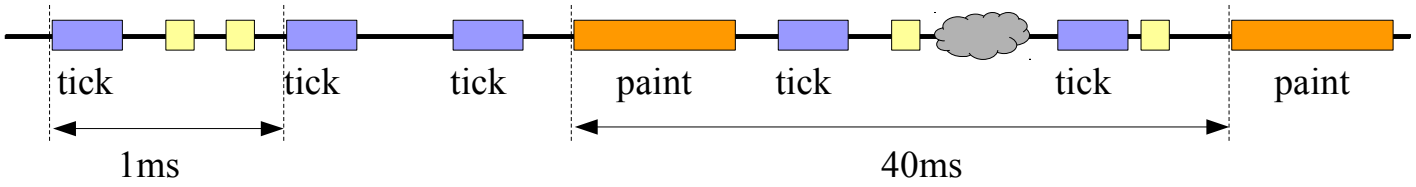
- A model
  - Your virtual world, it evolves as time passes
- A view
  - Rendering your world, 24 times per second
- A controller
  - Reacting to user inputs (keyboard or mouse)
  - Act upon the artefacts in your virtual world

A Game --- Execution Flow

- Single-threaded event-oriented execution flow



Timeline:

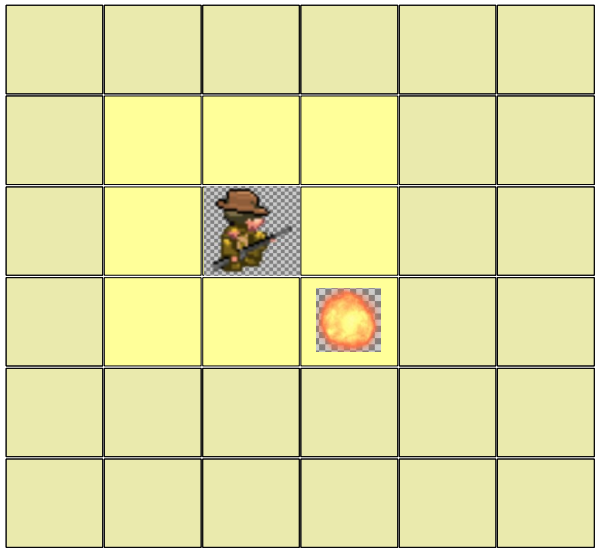


A Game --- The simulation of a virtual world

- A virtual world
  - A space
    - 2D or 3D space → 2D is much simpler
    - A space with things in it → players, robots, obstacles, etc.
  - Laws of physics
    - Gravity, momentum, etc.
    - Birth and decay, energy, etc.
    - Obstacles may be solid or can be passed through
- Passing time
  - How often should the time tick?
  - Every milliseconds? Less?
  - It depends on the speed of the physical phenomenons in your world…

Rendering

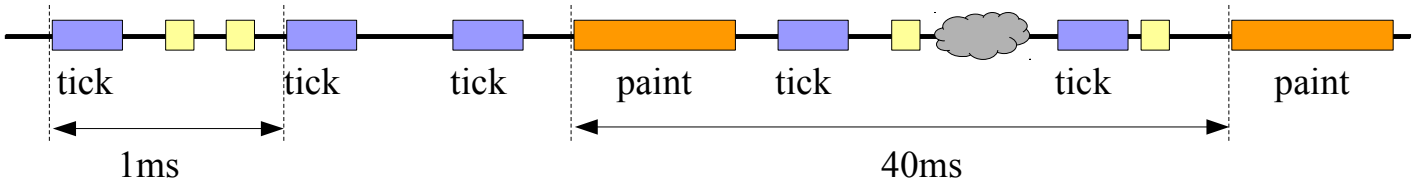
- Global view or view port?
  - Always render only the "*visible*" part of the map
  - You may decide to keep the entire map visible
- Render your world in 2D cells
  - To paint by browsing your model
  - Your model holds a 2D map, with cells
- Each cell: background + foreground
  - Background: color or image
  - Foreground: entities
- Paint entities
  - Shapes such as points, lines, rectangles, polygons
  - Wire-drawn or color-filled shapes
  - One image or a stack of images



Rendering

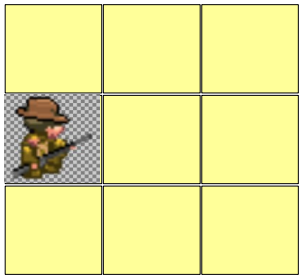
- Frames Per Second (FPS)
  - How many times do you render per second?
  - Movie → 24 fps (enough to be fluid)
- Rendering every 41 milliseconds (24fps)
  - You have 41ms for everything
  - Reacting to the tick event, every millisecond
  - Reacting to all GUI events
  - Rendering your model

Timeline:

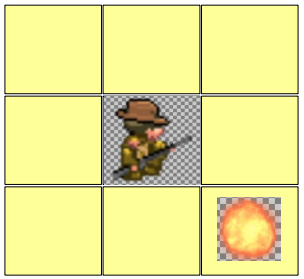


Rendering

- We use double buffering
  - A classical technique to avoid flickering
  - We have our own home-made double buffering
- Simple idea
  - Separate drawing and rendering
  - You draw in one buffer (an image)
  - The toolkit renders the other buffer
  - You swap the buffers at every cycle



Render this frame on the screen...



Draw your next frame here...

Controller

- Multi-modal interaction
  - Mouse events
    - Move movements (dx,dy) → 80 to 250 times per second
    - Move buttons (pressed,released,clicked)
  - Keyboard events
    - Key pressed, released, typed
    - Key code versus “character”
- Locate what is clicked
  - Maybe as simple as (x,y) → cell → entity
  - Watch out if you allow more than one entity per cell
  - Forward the events to the entity
  - Don’t forget enter/leave events that you will have to generate

Conclusion

- Read and understand the given framework
  - Experiment with it… with simple stuff…
- Understand event-oriented programming
  - It will take some time to getting use to it
  - Start simple also, experiment debugging
- You will use tracing
  - You may consider using a logger (log4j or java.util.Logger)
  - At the very least control logging via static boolean variables
- Organize your software development
  - Use the MVC modularity as a starting point
- Observe your game performance
  - Tick variations, FPS variations
  - Use jvisualvm