Basic concept of 2D game design and development

David Botzenhart, Kathrin Krisch

19th January 2007
Analysis
What makes a 2D Game?
Transformation of the Design Analysis into a Software Model.

Modeling
The Game class.
Animations and Sounds ("The moving pictures")

Events and Collision Detection
Events
Collision Detection
Good Bye
The Idea

Beside the fact that a good “programming” is not the only key to a successful game, we will focus on thinking of “how” to develop a game in a technical approach.

The most important thing, of course, is the idea, closely followed by good design (audio and video) and a stable realization. Keep in mind that games are made for entertainment. But also, games can be a source for powerful ideas...
Recognizable parts of a 2D game

What does appear in a game? Let us think about it...

▶ Screen.
Recognizable parts of a 2D game

What does appear in a game? Let us think about it...

▶ Screen.

▶ Sounds.
What makes a 2D Game?
Transformation of the Design Analysis into a Software Model.

Recognizable parts of a 2D game

What does appear in a game? Let us think about it...

- Screen.
  - Frame or Status Bar.

- Sounds.
Recognizable parts of a 2D game

What does appear in a game? Let us think about it…

▶ Screen.
  ▶ Frame or Status Bar.

▶ Game Window.

▶ Sounds.
Recognizable parts of a 2D game

What does appear in a game? Let us think about it...

▶ Screen.
  ▶ Frame or Status Bar.
    ▶ Credits (Points, Score).

▶ Game Window.

▶ Sounds.
Recognizable parts of a 2D game

What does appear in a game? Let us think about it...

- Screen.
  - Frame or Status Bar.
    - Credits (Points, Score).
    - Status.
  - Game Window.

- Sounds.
What does appear in a game? Let us think about it…

- Screen.
  - Frame or Status Bar.
    - Credits (Points, Score).
    - Status.
    - Chat Screen, Maps, etc.
  - Game Window.

- Sounds.
Recognizable parts of a 2D game

What does appear in a game? Let us think about it...

- Screen.
  - Frame or Status Bar.
    - Credits (Points, Score).
    - Status.
    - Chat Screen, Maps, etc.
  - Game Window.
    - Dynamic Objects (Items).

- Sounds.
Recognizable parts of a 2D game

What does appear in a game? Let us think about it...

- Screen.
  - Frame or Status Bar.
    - Credits (Points, Score).
    - Status.
    - Chat Screen, Maps, etc.
- Game Window.
  - Dynamic Objects (Items).
  - Cursor.

- Sounds.
Recognizable parts of a 2D game

What does appear in a game? Let us think about it...

- Screen.
  - Frame or Status Bar.
    - Credits (Points, Score).
    - Status.
    - Chat Screen, Maps, etc.
  - Game Window.
    - Dynamic Objects (Items).
    - Cursor.
    - Mouse Over Texts.
- Sounds.
Recognizable parts of a 2D game

What does appear in a game? Let us think about it...

▶ Screen.
  ▶ Frame or Status Bar.
    ▶ Credits (Points, Score).
    ▶ Status.
    ▶ Chat Screen, Maps, etc.
  ▶ Game Window.
    ▶ Dynamic Objects (Items).
    ▶ Cursor.
    ▶ Mouse Over Texts.
    ▶ Background.

▶ Sounds.
Recognizable parts of a 2D game (continued)
Transformation of the Design Analysis into a Software Model.

What do we need?

- A game class.

- Should be something like a thread.
- Should have access to graphic and/or audio interface.
- Should contain, control and care for all items.
- Should manage or delegate user inputs and events.
- Should be time based (game loop).
- Should contain the game logic.

- A basic item class.
- Should have a defined status (over time).
- Should have a position.
- Should have a dimension (area, for collision detection, etc.).
- Should be able to draw itself.
- Should play its sounds.
Transformation of the Design Analysis into a Software Model.

What do we need?

▶ A game class.
  ▶ Should be something like a thread.
Transformation of the Design Analysis into a Software Model.

What do we need?

- A game class.
  - Should be something like a thread.
  - Should have access to graphic and/or audio interface.

David Botzenhart, Kathrin Krisch
Basic concept of 2D game design and development
Transformation of the Design Analysis into a Software Model.

What do we need?

- A game class.
  - Should be something like a thread.
  - Should have access to graphic and/or audio interface.
  - Should contain, control and care for all items.
What makes a 2D Game?

Transformation of the Design Analysis into a Software Model.

What do we need?

▶ A game class.
  ▶ Should be something like a thread.
  ▶ Should have access to graphic and/or audio interface.
  ▶ Should contain, control and care for all items.
  ▶ Should manage or delegate user inputs and events.
Transformation of the Design Analysis into a Software Model.

What do we need?

- A game class.
  - Should be something like a thread.
  - Should have access to graphic and/or audio interface.
  - Should contain, control and care for all items.
  - Should manage or delegate user inputs and events.
  - Should be time based (game loop).
Transformation of the Design Analysis into a Software Model.

What do we need?

▶ A game class.
  ▶ Should be something like a thread.
  ▶ Should have access to graphic and/or audio interface.
  ▶ Should contain, control and care for all items.
  ▶ Should manage or delegate user inputs and events.
  ▶ Should be time based (game loop).
  ▶ Should contain the game logic.
Transformation of the Design Analysis into a Software Model.

What do we need?

- A game class.
  - Should be something like a thread.
  - Should have access to graphic and/or audio interface.
  - Should contain, control and care for all items.
  - Should manage or delegate user inputs and events.
  - Should be time based (game loop).
  - Should contain the game logic.

- A basic item class.
  - Should have a defined status (over time).
  - Should have a position.
  - Should have a dimension (area, for collision detection, etc.).
  - Should be able to draw itself.
  - Should play its sounds.
Transformation of the Design Analysis into a Software Model.

What do we need?

▶ A game class.
  ▶ Should be something like a thread.
  ▶ Should have access to graphic and/or audio interface.
  ▶ Should contain, control and care for all items.
  ▶ Should manage or delegate user inputs and events.
  ▶ Should be time based (game loop).
  ▶ Should contain the game logic.

▶ A basic item class.
  ▶ Should have a defined status (over time).
Transformation of the Design Analysis into a Software Model.

What do we need?

- A game class.
  - Should be something like a thread.
  - Should have access to graphic and/or audio interface.
  - Should contain, control and care for all items.
  - Should manage or delegate user inputs and events.
  - Should be time based (game loop).
  - Should contain the game logic.

- A basic item class.
  - Should have a defined status (over time).
  - Should have a position.
Transformation of the Design Analysis into a Software Model.

What do we need?

- A game class.
  - Should be something like a thread.
  - Should have access to graphic and/or audio interface.
  - Should contain, control and care for all items.
  - Should manage or delegate user inputs and events.
  - Should be time based (game loop).
  - Should contain the game logic.

- A basic item class.
  - Should have a defined status (over time).
  - Should have a position.
  - Should have a dimension (area, for collision detection, etc.).
What do we need?

- A game class.
  - Should be something like a thread.
  - Should have access to graphic and/or audio interface.
  - Should contain, control and care for all items.
  - Should manage or delegate user inputs and events.
  - Should be time based (game loop).
  - Should contain the game logic.

- A basic item class.
  - Should have a defined status (over time).
  - Should have a position.
  - Should have a dimension (area, for collision detection, etc.).
  - Should be able to draw itself.
What do we need?

- A game class.
  - Should be something like a thread.
  - Should have access to graphic and/or audio interface.
  - Should contain, control and care for all items.
  - Should manage or delegate user inputs and events.
  - Should be time based (game loop).
  - Should contain the game logic.

- A basic item class.
  - Should have a defined status (over time).
  - Should have a position.
  - Should have a dimension (area, for collision detection, etc.).
  - Should be able to draw itself.
  - Should play its sounds.
Transformation from Design Analysis to Software Model

Dynamic Items
Map
Status etc.

David Botzenhart, Kathrin Krisch
Basic concept of 2D game design and development
The Game class states.

- **stopped**
- **run**
- **play**
- **game loop**
- **initialize game components**
- **clear up components**

**Start**
- **stop**
The Game Class’s basic methods

::Game
GameItems[]

run()
paint()
The Games’ GameItem Collection

::Game
GameItems[]

run()
paint()

::GameItem
update(long timestamp)
The update iteration (loop)

Game

GameItem

run()
paint()

update(long timestamp)
The paint iteration (loop)

::Game
  GameItems[] :
    run()
    paint()

::GameItem
  update(long timestamp)
  paint(graphic context)
  ::GameItem
  update(long timestamp)

::Game
  paint(graphic context)
Time based global values (statics)

```
::Game
GameItems[]
static SINE
static SAW

run()
paint()

::GameItem
update( long timestamp)
::GameItem
update( long timestamp)
run()
::GameItem
update( long timestamp)
::Game
paint()
::GameItem
update( long timestamp)
GameItems[]
paint(graphic context)
static SINE
static SAW
```
Time based global values (cont’d)

```
::Game
GameItems[]
static SINE
static SAW
run()
paint()

::GameItem
update(long timestamp)
```

```
::Game
GameItems[]
static SINE
static SAW
run()
paint()

::GameItem
update(long timestamp)
```
Time based global values (cont’d)

### The Game class.
- Animations and Sounds ("The moving pictures")

#### Introduction

- Analysis
- Modeling
- Events and Collision Detection

#### Time based global values (cont’d)

<table>
<thead>
<tr>
<th>::Game</th>
</tr>
</thead>
<tbody>
<tr>
<td>GameItems[]</td>
</tr>
<tr>
<td>static SINE</td>
</tr>
<tr>
<td>static SAW</td>
</tr>
</tbody>
</table>

| run() |
| paint() |

```
run()
::Game
paint()
GameItems[]
static SINE
static SAW
```
Time based global values (cont’d)

::Game
GameItems[]
static SINE
static SAW
run()
paint()
The Gameltem as Container

::GameItem
*Array[] *Sounds
*Array[] *Animations

paint(graphic context)
update(long timestamp)

::Sound
playSound(Sound Context)

::Animation
getImage(frame)
getImage(timeIndex)
The GameItem as Container cont’d

::GameItem
- *Array[]* *Sounds*
- *Array[]* *Animations*

- `paint(graphic context)`
- `update(long timestamp)`

::Sound
- `playSound(Sound Context)`

::Animation
- `getImage(frame)`
- `getImage(timeIndex)`

Lightweight objects.
## Animation

<table>
<thead>
<tr>
<th>Animation</th>
</tr>
</thead>
<tbody>
<tr>
<td>image</td>
</tr>
<tr>
<td>framerate</td>
</tr>
<tr>
<td>getImage(frame)</td>
</tr>
<tr>
<td>getImage(timeIndex)</td>
</tr>
<tr>
<td>loadImage(FileName)</td>
</tr>
</tbody>
</table>

Basic concept of 2D game design and development

David Botzenhart, Kathrin Krisch
Animation (cont’d)

<table>
<thead>
<tr>
<th>Animation</th>
</tr>
</thead>
<tbody>
<tr>
<td>image</td>
</tr>
<tr>
<td>framerate</td>
</tr>
<tr>
<td>getImage(frame)</td>
</tr>
<tr>
<td>getImage(timeIndex)</td>
</tr>
<tr>
<td>loadImage(FileName)</td>
</tr>
</tbody>
</table>

David Botzenhart, Kathrin Krisch

Basic concept of 2D game design and development
Animation (cont’d)

<table>
<thead>
<tr>
<th>Animation</th>
</tr>
</thead>
<tbody>
<tr>
<td>image</td>
</tr>
<tr>
<td>framerate</td>
</tr>
<tr>
<td>getImage(frame)</td>
</tr>
<tr>
<td>getImage(timeIndex)</td>
</tr>
<tr>
<td>loadImage(FileName)</td>
</tr>
</tbody>
</table>

Frame

0 1 2 3 4 5 6 7
Animation (cont’d)

<table>
<thead>
<tr>
<th>Animation</th>
</tr>
</thead>
<tbody>
<tr>
<td>image</td>
</tr>
<tr>
<td>framerate</td>
</tr>
<tr>
<td>getImage(frame)</td>
</tr>
<tr>
<td>getImage(timeIndex)</td>
</tr>
<tr>
<td>loadImage(FileName)</td>
</tr>
</tbody>
</table>

anyAnimation.getImage(4)

Frame

0 1 2 3 4 5 6 7
Types of Events

- The Internal Events.

- Timer Event (internal).
- GameItem Event (shoots at other Item).

- External Events (System Events).
- Events caused by User (Input Device Event).
  - Mouse Event.
  - Keyboard Event.
  - Joystick etc.
- Other System Events.
  - Forced Repaint.
  - Forced Shutdown.
  - Set to Background, Timer Event (external) etc.
Types of Events

- The Internal Events.
  - Timer Event (internal).

David Botzenhart, Kathrin Krisch
Basic concept of 2D game design and development
Types of Events

- The Internal Events.
  - Timer Event (internal).
  - Gameltem Event (shoots at other Item).
Types of Events

- The Internal Events.
  - Timer Event (internal).
  - GameItem Event (shoots at other Item).
  - Collision Detection.

- External Events (System Events).
  - Events caused by User (Input Device Event).
    - Mouse Event.
    - Keyboard Event.
    - Joystick.
  - Other System Events.
    - Forced Repaint.
    - Forced Shutdown.
    - Set to Background, Timer Event (external) etc.
Types of Events

- The Internal Events.
  - Timer Event (internal).
  - Gameltem Event (shoots at other Item).
  - Collision Detection.

- External Events (System Events).
  - Events caused by User (Input Device Event).
    - Mouse Event.
    - Keyboard Event.
    - Joystick etc.
  - Other System Events.
    - Forced Repaint.
    - Forced Shutdown.
    - Set to Background, Timer Event (external) etc.
Types of Events

- The Internal Events.
  - Timer Event (internal).
  - GameItem Event (shoots at other Item).
  - Collision Detection.

- External Events (System Events).
  - Events caused by User (Input Device Event).
Types of Events

- The Internal Events.
  - Timer Event (internal).
  - GameItem Event (shoots at other Item).
  - Collision Detection.

- External Events (System Events).
  - Events caused by User (Input Device Event).

- Other System Events.
Types of Events

- The Internal Events.
  - Timer Event (internal).
  - GameItem Event (shoots at other Item).
  - Collision Detection.

- External Events (System Events).
  - Events caused by User (Input Device Event).
    - Mouse Event.
    - Keyboard Event.
    - Joystick etc.
  - Other System Events.
Types of Events

▶ The Internal Events.
  ▶ Timer Event (internal).
  ▶ GameItem Event (shoots at other Item).
  ▶ Collision Detection.

▶ External Events (System Events).
  ▶ Events caused by User (Input Device Event).
    ▶ Mouse Event.
    ▶ Keyboard Event.
    ▶ Joystick etc.
  ▶ Other System Events.
    ▶ Forced Repaint.
    ▶ Forced Shutdown.
    ▶ Set to Background, Timer Event (external) etc.
Event processing

System

Any::GameItem

::Game

::GameWindow

User Input
Collision detection

```
::Game
GameItems[]

checkForCollision()
eventHandler(Event)
run()
paint()
```

```
::GameItem
isCollided(::GameItem)
```

```
::GameItem
isCollided(::GameItem)
```

David Botzenhart, Kathrin Krisch
Basic concept of 2D game design and development
Collision detection (cont’d)

This basic algorithm would have the complexity:

\[ C = \frac{n \times (n - 1)}{2} \]

\[ O(n) = n^2 \]

...number of GameItems which may collide. Of course this can be improved, but that is a different story...
Thank you for your participation...