# Gamestates, Buttons und SD-Karte

## u23 2012

andy, gordin, ike

Chaos Computer Club Cologne e.V.
http://koeln.ccc.de

Cologne
2012-11-12



andy <andy@koeln.ccc.de>, gordin <gordin@koeln.ccc.de>, ike <ike@tura-home.de>

# Was sind Gamestates

- Zur Strukturierung von Spielen
- Aufgebaut als Stack (in unserer Library)
- States kennen den vorherigen State
- States nicht mehrmals in den Stack packen!
- Haben bei uns keinen State -> globale Variablen
- Funktion: State wechseln und zum vorherigen State zurückkehren
- Beispiel:
- Hauptmenü -> (wechsel) Optionen -> (zurück) Hauptmenü
- Hauptmenü -> (wechsel) Spiel -> (wechsel) Highscore -> (zurück) Spiel -> (zurück) Hauptmenü



andy <andy@koeln.ccc.de>, gordin <gordin@koeln.ccc.de>, ike <ike@tura-home.de>

# Unsere Gamestates

- Unsere Gamestates: keine Daten, nur Funktionspointer

```
Gamestate InitState = { Init, OnEnter, OnLeave, Update, Draw };
```

- Init wird 1 mal ausgeführt, beim ersten Betreten des States
- OnEnter wird jedes mal ausgeführt, beim Betreten des States
- OnLeave wird jedes mal ausgeführt, beim Wechseln des States
- Update und Draw werden jeden Frame nacheinander ausgeführt
- Init, OnEnter und OnLeave bekommen Pointer auf den Gamestate
- Update bekommt Zeit seit letztem Aufruf in 10ms
- Draw bekommt einen Pointer auf das Bitmap, was über VGA ausgegeben wird



andy <andy@koeln.ccc.de>, gordin <gordin@koeln.ccc.de>, ike <ike@tura-home.de>

# Gamestates wechseln

- Zu neuem State wechseln:

```
int ChangeState(Gamestate *state);
```

- Zum vorherigen State zurückkehren:

```
void ExitState(void);
```

- Wichtig: ChangeState und ExitState nur in Update aufrufen!
- Beispiel mit 2 States in firmware/gamestate_test (Game.c und Red.c)



andy <andy@koeln.ccc.de>, gordin <gordin@koeln.ccc.de>, ike <ike@tura-home.de>

# Onboard Buttons

- Definiert in PushButtons.h und PushButtons.c
- Header automatisch über Game.h inkludiert
- Hardwarestate komplett asynchron ausgelesen über Timer Interrupt
- Eine Funktion um zuletzt bekannten State abzuholen:

```
pushbutton_button_state_t GetPushbuttonState(void);
```



andy <andy@koeln.ccc.de>, gordin <gordin@koeln.ccc.de>, ike <ike@tura-home.de>

# Onboard Buttons

- pushbutton_button_state_t:

```
typedef struct {
uint16_t Up : 1;
uint16_t Down : 1;
uint16_t Left : 1;
uint16_t Right : 1;
uint16_t A : 1;
uint16_t B : 1;
uint16_t User : 1;
} pushbutton_button_state_t;
```

- Beispiel:

```
pushbutton_button_state_t buttons = GetPushbuttonState();
if(buttons.Up)
moveUp();
else if(buttons.Down)
moveDown();
```



andy <andy@koeln.ccc.de>, gordin <gordin@koeln.ccc.de>, ike <ike@tura-home.de>

# SNES Controller

- Definiert in SNES.h und SNES.c
- Header automatisch über Game.h inkludiert
- Hardwarestate komplett asynchron ausgelesen über Timer Interrupt
- Eine Funktion um zuletzt bekannten State abzuholen:

```
snes_button_state_t GetControllerState1(void);
snes_button_state_t GetControllerState2(void);
```



andy <andy@koeln.ccc.de>, gordin <gordin@koeln.ccc.de>, ike <ike@tura-home.de>

# SNES Controller

- snes_button_state_t:

```
typedef union {
struct {
uint16_t Reserved : 4;
uint16_t R : 1;
uint16_t L : 1;
uint16_t X : 1;
uint16_t A : 1;
uint16_t Right : 1;
uint16_t Left : 1;
uint16_t Down : 1;
uint16_t Up : 1;
uint16_t Start : 1;
uint16_t Select : 1;
uint16_t Y : 1;
uint16_t B : 1;
} buttons;
uint16_t raw;
} snes_button_state_t;
```



andy <andy@koeln.ccc.de>, gordin <gordin@koeln.ccc.de>, ike <ike@tura-home.de>

# SNES Controller

- Beispiel:

```
snes_button_state_t snes = GetControllerState1();
if(snes.buttons.Up)
moveUp();
else if(snes.buttons.Down)
moveDown();
```



andy <andy@koeln.ccc.de>, gordin <gordin@koeln.ccc.de>, ike <ike@tura-home.de>

# Dateisystem

- Beispielcode in gametest
- Benötigte Header: game/Filesystem.h
- Danach einmal *InitializeFilesystem()* aufrufen
- Deaktivieren wieder mit *DeinitializeFilesystem()*
- Code nutzt die erste Partition der SD-Karte
- Muss mit FAT12, FAT16 oder FAT32 formatiert sein



andy <andy@koeln.ccc.de>, gordin <gordin@koeln.ccc.de>, ike <ike@tura-home.de>

# API

- API ist normale POSIX-API:
- *fopen()*, *fclose()*, *fread()*, *fwrite()*, *fseek()*, *ftell()*
- Wie genau die Aufrufe aussehen seht ihr im gametest
- In der VM existieren manpages dazu:
- Zum Beispiel *man fopen* auf der Konsole eingeben und lesen
- Beschränkung in unserer Library: Max 3 gleichzeitig geöffnete Dateien (Filepointer)



andy <andy@koeln.ccc.de>, gordin <gordin@koeln.ccc.de>, ike <ike@tura-home.de>

# Doubly Linked List

- void listInsert(list *l, void *s);
- void listAppend(list *l, void *s);
- void listInsertBefore(list *l, list_el *item, void *s);
- void listInsertAfter(list *l, list_el *item, void *s);
- void listRemove(list *l, list_el *item);
- void listRemoveByValue(list *l, void *s);
- void listRemoveAllByValue(list *l, void *s);



andy <andy@koeln.ccc.de>, gordin <gordin@koeln.ccc.de>, ike <ike@tura-home.de>

# Doubly Linked List

- #include <list.h>
- Initialisieren
- list *name = &(list) { };
- Items hinzufuegen:
- listInsert(name, daten);



andy <andy@koeln.ccc.de>, gordin <gordin@koeln.ccc.de>, ike <ike@tura-home.de>

# SpriteEngine

- #include <SpriteEngine.h>
- Verwaltet Bilder zusammen mit Koordinaten
- Kann alles auf einmal zeichnen
- void SpriteEngine_draw(Bitmap *surface, int xo, int yo)



andy <andy@koeln.ccc.de>, gordin <gordin@koeln.ccc.de>, ike <ike@tura-home.de>

# Uebersicht

- Ganzer code im Moment noch in libgraphics
- #include <TiledMap.h>
- Neue spacecraft-version
- github.com/cccc/U23-Library newspace



andy <andy@koeln.ccc.de>, gordin <gordin@koeln.ccc.de>, ike <ike@tura-home.de>

# Featuers

- Objekte mit Koordinaten
- Bewegung
- Animation
- Collision detection
- Event callbacks
- Isometrie (noch nicht)



andy <andy@koeln.ccc.de>, gordin <gordin@koeln.ccc.de>, ike <ike@tura-home.de>

# Grundlegendes

- Neue Map erzeugen:
- TiledMap* TiledMap_init(int sizeX, int sizeY, uint8_t tileSize, TileInfo *tileInfo);
- sizeX / sizeY: Groesse der Karten (Kacheln)
- tileSize: Groesse der Bitmap der Kacheln in Pixeln
- tileInfo: Ein array mit Informationen ueber die Kacheln
- (Bitmaps, Kollision?)



andy <andy@koeln.ccc.de>, gordin <gordin@koeln.ccc.de>, ike <ike@tura-home.de>

# Grundlegendes

- Loop Calls (jedes mal in den Schleifen aufrufen):
- void TiledMap_update(TiledMap *map, uint32_t delta);
- Updatet Animationen, Bewegung, Collision Detection, ...
- void TiledMap_draw(Bitmap *surface, TiledMap *map, int xo, int yo);
- Zeichnet einmal alles



andy <andy@koeln.ccc.de>, gordin <gordin@koeln.ccc.de>, ike <ike@tura-home.de>

# Objekte

- MapObject
- x, y: Position (Koordinaten in PIXEL_RESOLUTION pro Pixel)
- bitmap: Aktuelles Bild
- sizeX, sizeY: Rechteckgroesse (ebenfalls mit PIXEL_RESOLUTION)
- collision: Wie kollidiert das Objekt?
- moving: Informationen ueber aktuelle Bewegung
- animation: Informationen ueber aktuelle Animation



andy <andy@koeln.ccc.de>, gordin <gordin@koeln.ccc.de>, ike <ike@tura-home.de>

# Bewegung

- void MObj_moveDirection(MapObject *obj, int velX, int velY, bool collision);
- void MObj_moveGravity(MapObject *obj, int velX, int velY, bool collision);
- void MObj_moveTo(MapObject *obj, int tx, int ty, int speed, bool collision);
- void MObj_moveForced(MapObject *obj, int velX, int velY, int forceX, int forceY, bool collision);
- void MObj_cancelMovement(MapObject *obj);
- Callback: moving->targetReached



andy <andy@koeln.ccc.de>, gordin <gordin@koeln.ccc.de>, ike <ike@tura-home.de>

# Animation

- MapObject.animation
- Bekommt Array von Bitmaps
- und Animationsart
- ANIM_NORMAL
- ANIM_REVERSED
- ANIM_REPEAT
- ANIM_REPEAT_REVERSED



andy <andy@koeln.ccc.de>, gordin <gordin@koeln.ccc.de>, ike <ike@tura-home.de>

# Collision detection

- Callback bei Kollision:
- moving->onCollision
- Oder manuell:
- #include <Collision.h>
- oder:
- bool MObj_collisionMObj(MapObject *obj, MapObject *target);



andy <andy@koeln.ccc.de>, gordin <gordin@koeln.ccc.de>, ike <ike@tura-home.de>

# Hacken

**Hackt Spiele!**



andy <andy@koeln.ccc.de>, gordin <gordin@koeln.ccc.de>, ike <ike@tura-home.de>