中級プログラミング演習コンテストサンプル
LCDゲーム作成キット

三重大学　情報工学科　大野 和彦　ohno@arch.info.mie-u.ac.jp

発表の内容

- 背景・概要
  - LCDゲームとは
  - このキットで何ができるか
- 本キットによるゲーム作成手順
- 今後の課題

LCDゲームとは

LCDゲーム

- 定義はあいまいだが電子ゲーム機の一種であり、一般的に
  - LCD(液晶画面)を表示装置として用いる
  - ソフトウェアはハード的に組み込み(遊べるゲームは固定)
  - 携帯型ゲーム機

  を指すことが多い。
- 代表例としてGAME&WATCH(任天堂、1980-1991)が挙げられる。同社の後のGAMEBOYやSwitchなどと比べると
  - 表示装置はモノクロ液晶かつセグメント方式
    - ドットで任意の絵を表現するドットマトリクス方式はまだ実現できていない
    - 今日の電卓やデジタル時計と同様、固定パーツをON/OFFする方式
  - ゲームプログラムは交換できない
  - 当時のコンピュータのスペック上、ゲーム内容は単純

セグメント方式のLCD(液晶表示装置)

- 1枚の液晶パネル上に、絵のパーツとなるセグメントを形成
  - セグメント毎に表示・非表示を変えられる
  - 7本の線分の組み合わせで数字を表現：電卓など
  - ゲーム機の場合、キャラやスコアなどの表示をこれで行う
- 制約
  - パネルは一枚しか使えない(ステージで絵を変えたりできない)
  - セグメントは互いに接触したり重なったりできない
  - 色は表現できない
  - 背景・前景として固定のカラー絵は使える

このキットで何ができるか

PC上でのLCDゲーム再現

- LCDゲームをPC上で再現したい
  - 実際にやっている人もいる
- 表示装置は現代のグラフィック機能が使える
  - セグメントに当たる画像をそれぞれ用意して表示すればよい
  - ゲームプログラミング上、オリジナルと同様に「セグメントのON/OFF」で制御したい
  - セグメントの作成や制御機構などをゼロから作るのは面倒

- 共通作業を補助ツールやライブラリ関数などに吸収
- ユーザは実装するゲーム固有の要素(パネル・セグメントのデザイン、キャラの挙動やゲームルール)のみ作るようにしたい

本キットが提供するもの

- lcd_convert_img
  - フラグメント(セグメント)データ生成ツール
  - 液晶パネルを1枚絵として用意すると、フラグメント画像や位置情報、画像登録コードなどを自動生成する
- lcd.c, lcd.h
  - LCDゲーム作成用のランタイムライブラリ
  - ゲームの基本エンジン
    - キー入力の自動チェック
    - 登録されたユニット(キャラ)の管理とタイマによるコールバック処理
  - 描画機能
    - 表示ONのフラグメントのみ描画、背景やゲーム機の描画

ユーザが用意するもの

- LCDパネル(と背景)の一枚絵
- ゲーム固有のコード
  - ゲームオーバー条件やスコア取得、難易度上昇などのルール周り
  - 各ユニットの構造体定義と行動・表示アルゴリズム
    - 一定時間ごとに自動コールバックされるので、「呼ばれたら1ステップ分の行動を行う」などの関数を作ればよい
    - 表示関数はユニットの位置などに合わせてフラグメントに表ON/OFF設定すればよい

本キットによるLCDゲーム作成手順

ゲーム作成の流れ

今回のサンプルゲームTREEで説明

1. ゲームの内容を考える
2. LCDパネル(と背景)画像を作成(xpm形式に変換)
3. lcd_convert_imgを使ってフラグメントデータを生成
4. ゲームのコードを書く
5. キットのライブラリ、3で生成したファイルと共にコンパイル

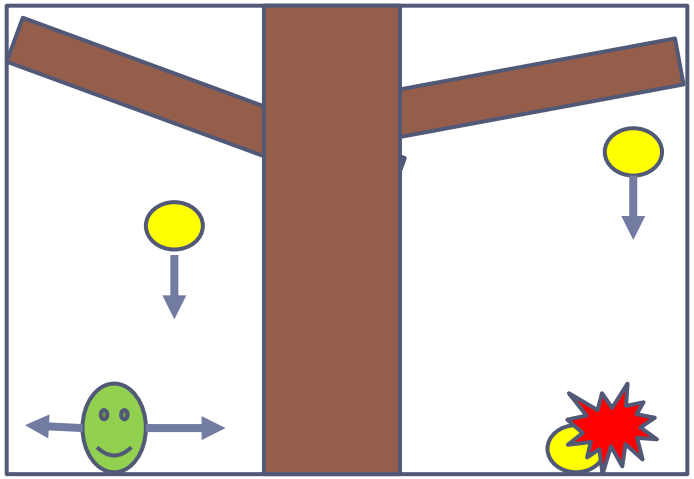
ゲーム作成の流れ

今回のサンプルゲームTREEで説明

1. ゲームの内容を考える
2. LCDパネル(と背景)画像を作成(xpm形式に変換)
3. lcd_convert_imgを使ってフラグメントデータを生成
4. ゲームのコードを書く
5. キットのライブラリ、3で生成したファイルと共にコンパイル

ゲームのデザイン

- 上から降ってくるアイテムをキャッチするゲーム
  - 自機は左右に移動する
  - アイテムはランダムなX座標に出現し、そのまま真下に落ちる
  - アイテムが地面に到達したらミス
  - 自機の高さでアイテムと時期が同じX座標なら自動でキャッチする
- LCDゲームなので
  - 自機・アイテムは重ねられない



ゲーム作成の流れ

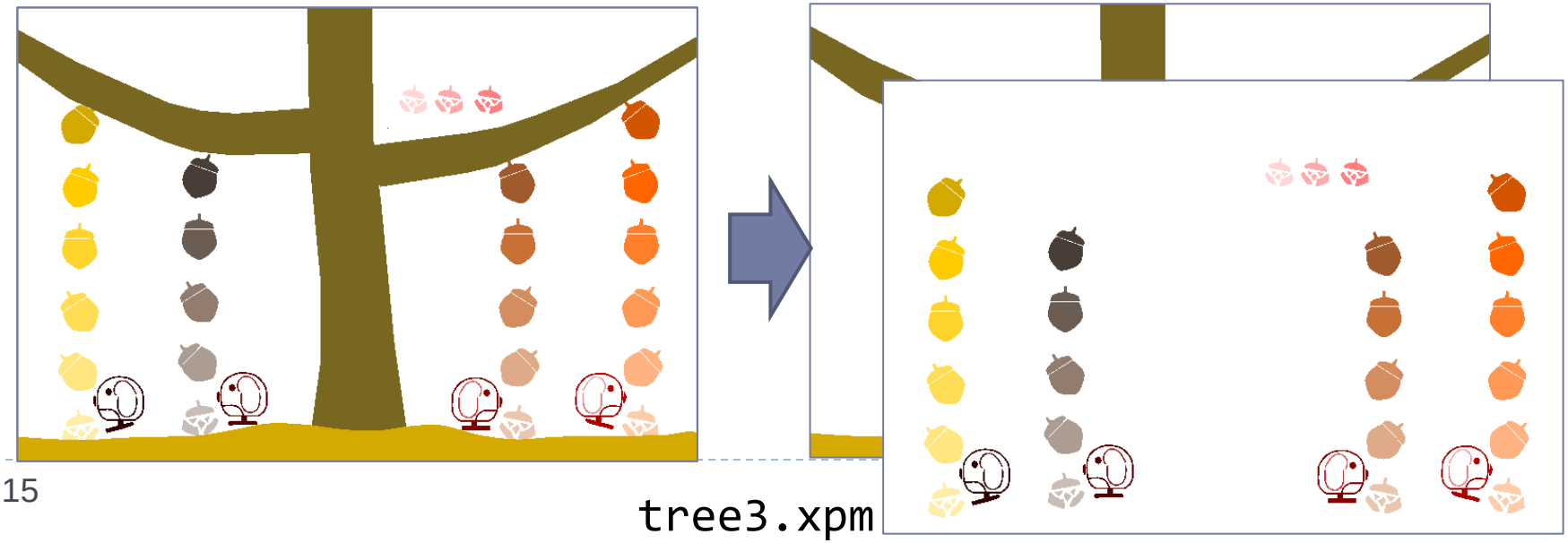
今回のサンプルゲームTREEで説明

1. ~~ゲームの内容を考える~~
2. LCDパネル(と背景)画像を作成(xpm形式に変換)
3. lcd_convert_imgを使ってフラグメントデータを生成
4. ゲームのコードを書く
5. キットのライブラリ、3で生成したファイルと共にコンパイル

画像の作成

- レイヤー機能のあるお絵かきソフトの使用を推奨
  - (ドロー系、ペイント系どちらでもよい)
- 今回はドロー系のInkscapeを使用
  - キャラはコピーし、回転などで個別に調整
  - 各位置毎に異なる色を設定
  - 背景とキャラを別レイヤーで作成し別々に出力

tree_bg.xpm



tree3.xpm

ゲーム作成の流れ

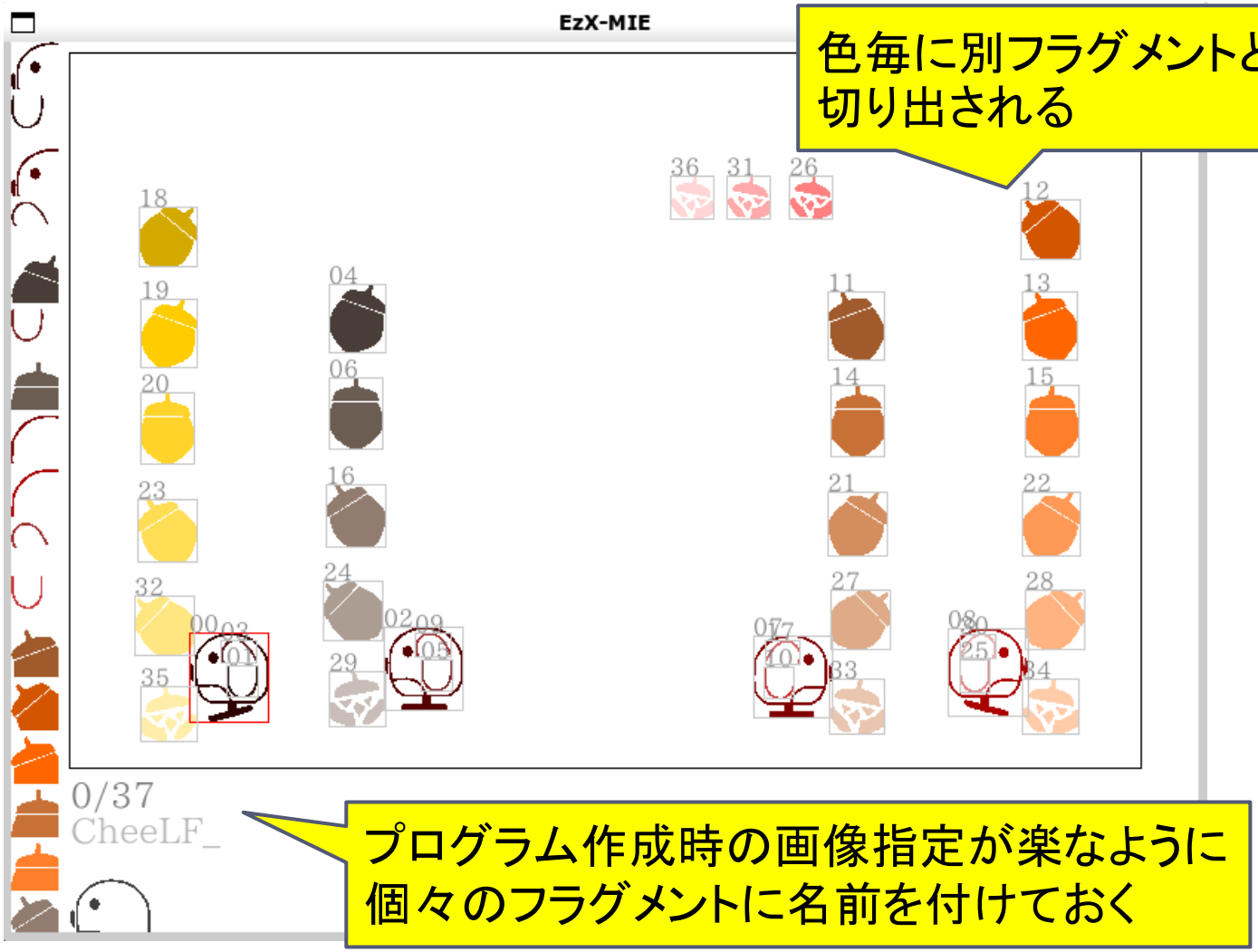
今回のサンプルゲームTREEで説明

1. ~~ゲームの内容を考える~~
2. ~~LCDパネル(と背景)画像を作成(xpm形式に変換)~~
3. lcd_convert_imgを使ってフラグメントデータを生成
4. ゲームのコードを書く
5. キットのライブラリ、3で生成したファイルと共にコンパイル

フラグメントデータの作成手順

- (画像形式の変換)
  - 今回はWindows上のInkscapeを使用した
    - 生成した画像をWSL上のLinux環境にコピーして使用
    - Inkscapeはxpm形式を出力できないのでpng形式で出力し、Linux側でImagemagickのconvertコマンドを使用して変換
- `./lcd_convert_img sample/tree3.xpm`
  でツールを起動し、GUI上で
  - 自動認識されたフラグメントにそれぞれ名前を付ける
  - フラグメントデータを出力させる
    - CTRL+Sで`tree3.dat`に設定情報を保存
    - CTRL+Gでフラグメントデータをファイルとして生成する

lcd_convert_imgの実行画面



生成されたフラグメントデータ(1)

以下のファイルが生成される

- tree3.h

```
#define LCD_FRAGMENT_NUM 37
typedef enum {
 LCD_fn_CheeLF, …
} LCD_fragment_name_t;
```

各フラグメントに
名前を付けるための
列挙体定義

- tree3.c

```
#include"tree3.h"
#include"tree3/CheeLF.xpm"
…
LCD_fragment_info_t LCD_fragment_infos[] = {
  {80, 390, CheeLF_xpm}, …
};
```

フラグメントxpmの読み込み
lcd.がEzImageへ登録し表
示するための情報配列

生成されたフラグメントデータ(2)

- CheeLF.xpmなど

```
/* XPM */static char *CheeLF_xpm[] = {
"54 61 2 1",
". c None",
"@ c #404040",
".........................@@.........................",
"...................@@@@@@@@@@@@@@@@...................",
"...............@@@@@@.........@@@@@@...............",
"..............@@@@...................@@@@.............",
"............@@@@.....................@@@@...........",
"...........@@@.........................@@@.........",
"..........@@...........................@@.........",
. . .
```

- 元画像から同色部分を最小矩形領域として切り出す
- 黒色の2色画像にする

ゲーム作成の流れ

今回のサンプルゲームTREEで説明

1. ~~ゲームの内容を考える~~
2. ~~LCDパネル(と背景)画像を作成(xpm形式に変換)~~
3. ~~lcd_convert_imgを使ってフラグメントデータを生成~~
4. ゲームのコードを書く
5. キットのライブラリ、3で生成したファイルと共にコンパイル

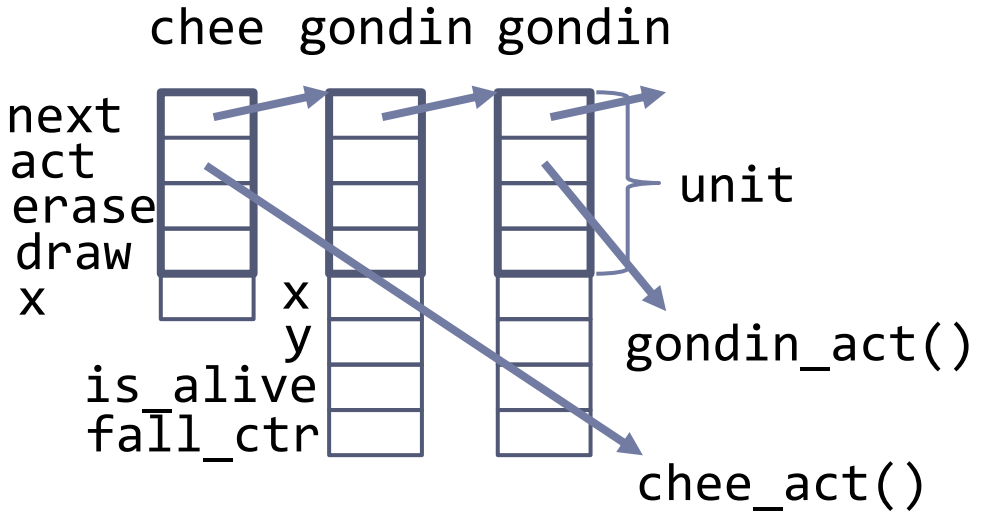
ユーザコードの一部(1) main関数

- LCD_で始まる関数・手数などは本キットが提供するもの
- LCD_init内で画像の登録、タイマの設定などは自動で行う
- initでユニットの生成・登録やスコアなどゲーム固有情報の設定などを行う
- EzEventLoop()呼出し後はコールバック関数で状況に応じた処理を行う

```
int main(void){
  LCD_init(720, 480);
  LCD_set_bg(tree_bg);
  init();
  EzEventLoop();
  LCD_finalize();
  return 0;
}
```

ユーザコードの一部(2) ユニット定義

```
typedef struct {
  LCD_unit_t unit;
  int x;
} u_chee_t;
typedef struct {
  LCD_unit_t unit;
  int x, y;
  int is_alive;
  int fall_ctr;
} u_gondin_t;
. . .
void init(void){
  chee = LCD_CreateUnit(u_chee_t);
  chee->unit.on_act = chee_act;
  chee->unit.on_erase = chee_erase;
  chee->unit.on_draw = chee_draw;
  chee->x = 1;
}
```



※簡略化してあるため実際のコードとは少し異なる

ユーザコードの一部(3) コールバック関数

```
LCD_fragment_name_t chee_fragments[4] ={
  LCD_fn_CheeLF, LCD_fn_CheeLN, LCD_fn_CheeRN, LCD_fn_CheeRF
};
void chee_act(LCD_unit_t *unit){
  unit_chee_t *chee = (unit_chee_t *)unit;
  if (game_state == gs_title) return;
  if (LCD_is_btn_pushed(LCD_btn_left) && chee->x > 0){
    chee->x--;
  }
  else if (LCD_is_btn_pushed(LCD_btn_right) && chee->x < 3){
    chee->x++;
  }
}
void chee_draw(LCD_unit_t *unit){
  unit_chee_t *chee = (unit_chee_t *)unit;
  LCD_show_fragment(chee_fragments[chee->x]);
}
```

ゲーム作成の流れ

今回のサンプルゲームTREEで説明

1. ~~ゲームの内容を考える~~
2. ~~LCDパネル(と背景)画像を作成(xpm形式に変換)~~
3. ~~lcd_convert_imgを使ってフラグメントデータを生成~~
4. ~~ゲームのコードを書く~~
5. キットのライブラリ、3で生成したファイルと共にコンパイル

コンパイル

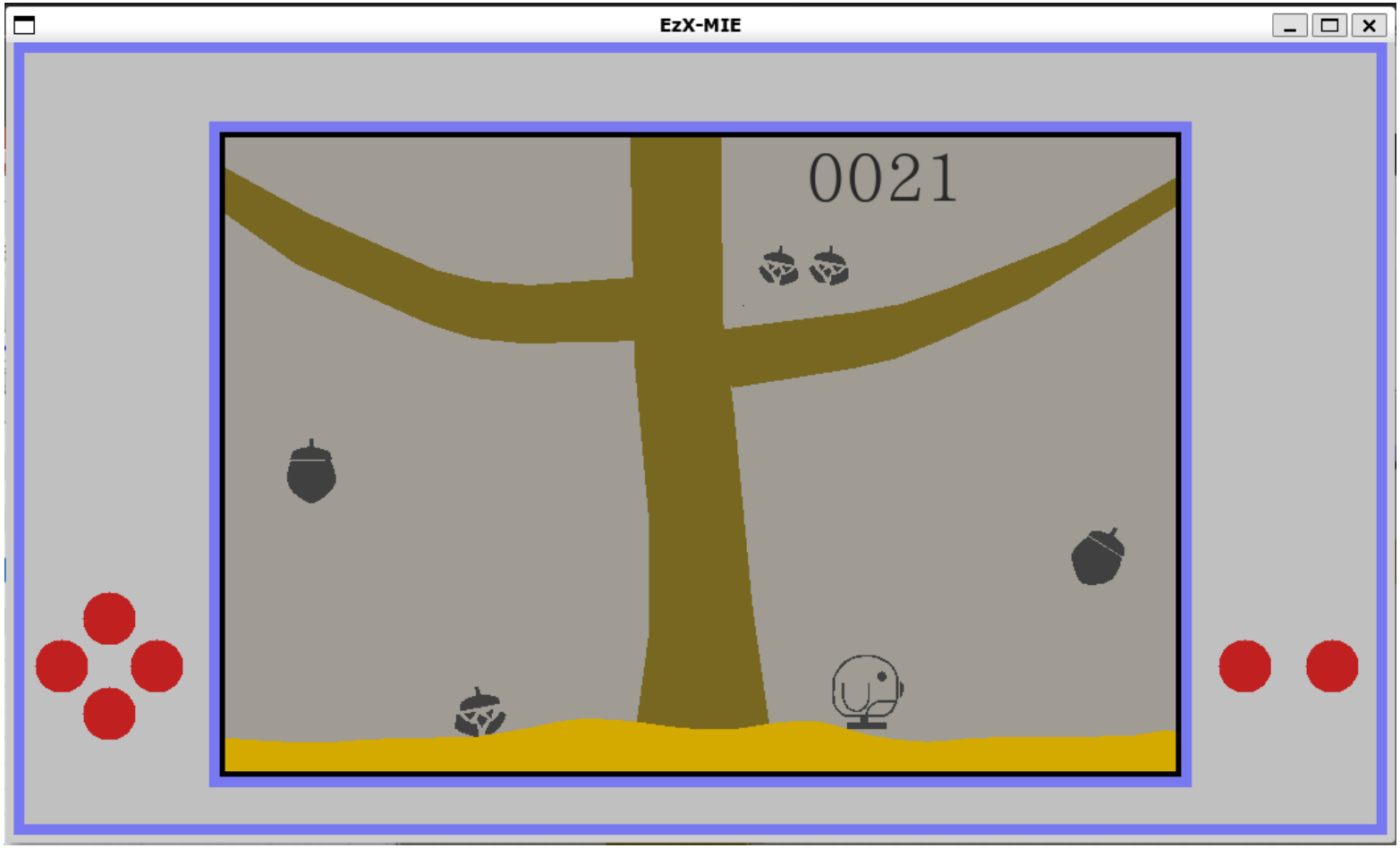
※ダウンロードしたファイルでビルドする場合、
make tree_game
と入力すればtree_gameが生成される

```
ezcc –Wall -lEzGraph tree_game.c tree3.c lcd.c
rnd.c –o tree_game
```

- tree_game.c, tree_game.h
  - TREEのプログラムコード
- tree3.c, tree3.h
  - lcd_convert_imgが自動生成したコード
- lcd.c, lcd.h
  - 本キットが提供するライブラリコード

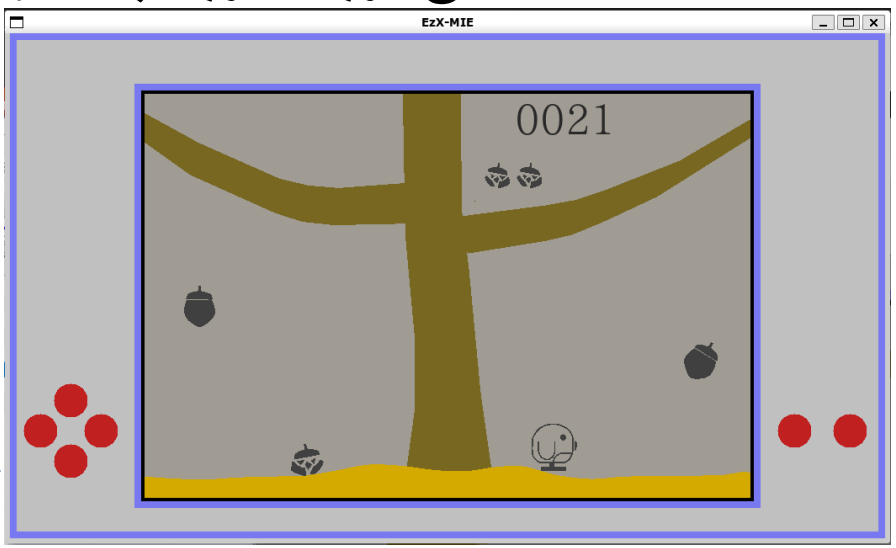
実行

- `./tree_game`



サンプルゲームの遊び方

- スペースキーでゲーム開始
- 左右カーソルキーで自キャラを操作し、上から降ってくる実をキャッチする
  - 実は地面に到達すると砕けて1ミスになる
  - 地面に到達する一つ前の高さの時に、自キャラが横にいると自動的にキャッチする
- 一定個数キャッチすると内部レベルが上がる
  - 実の出現頻度が上がる
  - 一度に複数出現する
- 内部レベルは最大になるとレベル0に戻る



今後の課題

- 機能の追加
  - 7セグメントを用いた数値表示機能
  - ゲーム機筐体のカスタマイズ機能(スキン機能?)
  - ゲームエンジンに一時停止やスローなど補助機能を追加
- サンプルゲームの追加