

ORGANIZACIJA RAČUNALNIKOV

2. Domača naloga - STM32H750B

Test reakcijskega časa

Test reakcijskega časa

Za 2. Domačo nalogo sem se odločil narediti test reakcijskega časa. Test reakcijskega časa postavi uporabnika pred izziv, ki testira njegove sposobnosti.

Uporabnik mora v najkrajšem času klikniti na tarčo in prikaže se mu reakcijski čas v milisekundah.

Ker ima plošča STM32H750B Development Kit zaslon na dotik, sem se odločil, da to izkoristim. Aplikacijo sem naredil s pomočjo knjižnice TouchGFX.

Za začetek moramo namestiti programe:

TouchGFX designer:

<https://www.st.com/en/development-tools/touchgfxdesigner.html>

STM32 Cube IDE:

<https://www.st.com/en/development-tools/stm32cubeide.html>

STM32 Cube Programmer:

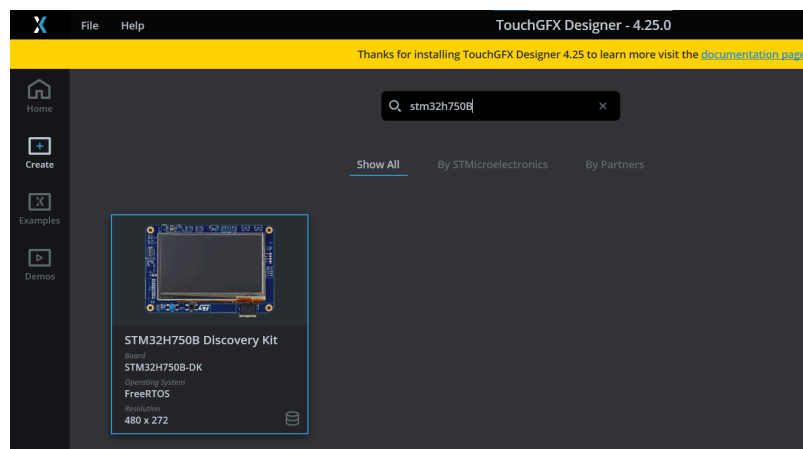
<https://www.st.com/en/development-tools/stm32cubeprog.html>

STM32 Cube Programmer nam omogoča brisanje in nalaganje programov v pomnilnik.

STM32 Cube IDE je razvojno okolje, v katerem lahko pišemo kodo. (Jaz sem uporabil tudi Visual Studio Code)

TouchGFX designer pa je knjižnica (ali pa v tem primeru kar samostojna aplikacija) s katero lahko implementiramo aplikacije s pomočjo uporabniškega vmesnika.

Za začetek ustvarimo nov projekt in določimo strojno opremo. V našem primeru je to **STM32H750BDK**:



V programu Photoshop sem pripravil ikone za tarče, ki sem jih uporabil v aplikaciji. Zaradi varčevanja prostora sem vsako sliko stisnil, tako, da vsaka slika zavzame le od 7 - 20 KB



Aplikacija je razdeljena na 3 zaslone:

Zaslon 1:

To je domači zaslon, ki se prikaže takoj ob začetku aplikacije.

Na njem so trije gumbi:

Prva igra - Kako hitro lahko zadanete tarčo?

Druga igra - Koliko tarč lahko zadanete v 30 sekundah?

Težavnost - spreminja velikost tarč.

Ob pritisku na prvi gumb se izvede interakcija, ki zamenja zaslon iz prvega na drugega.

Ob pritisku na drugi gumb se izvede interakcija, ki zamenja zaslon iz prvega na tretjega.

Ob pritisku na gumb za težavnost se shrani true/false vrednost in se prenese na drug zaslon.

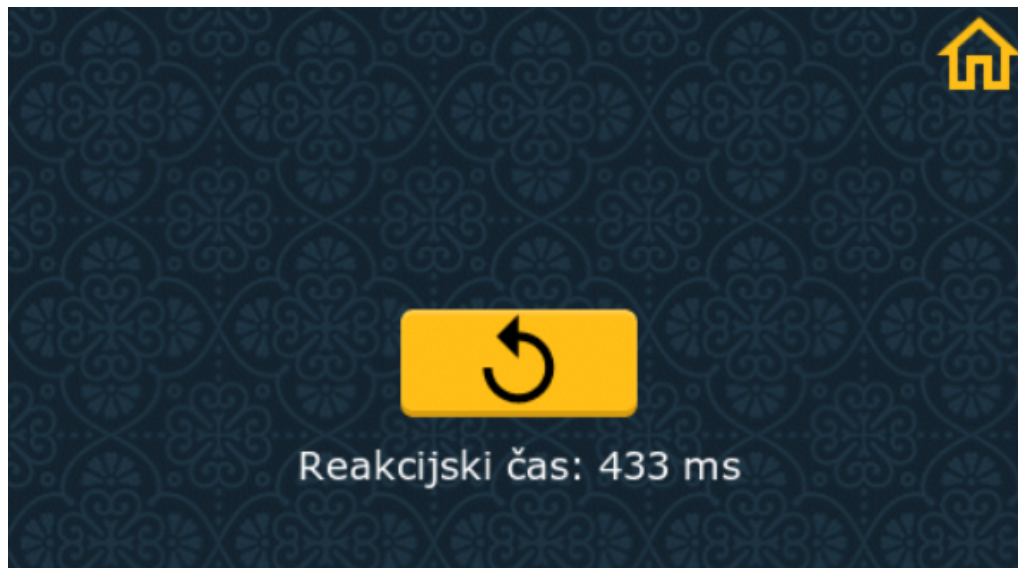


Zaslon 2:

Najprej se pojavi prazno ozadje. Po naključno med 3 in 5 sekundami se na naključnem mestu pojavi tarča.

Ko uporabnik klikne na tarčo, ta izgine, pojavijo pa se gumb za ponovno igranje, gumb za domov in besedilo, ki prikazuje koliko časa v ms je uporabnik potreboval, da je zadel tarčo.

Gumb domov vrne uporabnika na prvo stran, gumb za ponovno igranje pa počisti zaslon (razen ozadja), ponastavi vrednosti in znova čaka med 3 in 5 sekunde, da prikaže novo tarčo.



Zaslon 3:

Poleg praznega ozadja na vrhu opazimo tudi številko 30. Ta številka ponazarja 30 sekund. Ko se znova med 3 in 5 sekundami prikaže tarča, se začnejo odštevati sekunde.



Tokrat, namesto da bi po kliku tarče čakali na novo se naslednja takoj prikaže na naključnem mestu.

Ko se čas odšteje do 0, se prikaže gumb za domov, gumb za ponovno igranje in besedilo, ki pravi:

Zadeli ste [število_tarč] tarč.

Povprečen reakcijski čas na tarčo: [povprečen_reakcijski_cas] ms.



Koda:

.cpp datoteke vsebujejo implementacijo c++ kode, .hpp datoteke pa hranijo spremenljivke in funkcije, ki jih uporabljajo .cpp datoteke.

TouchGFX že samodejno generira kodo, ki je potrebna za delovanje aplikacije. Ta koda je *screen_numberViewBase.hpp* in *screen_numberViewBase.cpp* in je omogočena le za branje. Če želimo spreminjati elemente, moramo to storiti v *screen_numberView.cpp*.

Screen1View.hpp:

```
#ifndef SCREEN1VIEW_HPP
#define SCREEN1VIEW_HPP

#include <gui_generated/screen1_screen/Screen1ViewBase.hpp>
#include <gui/screen1_screen/Screen1Presenter.hpp>

class Screen1View : public Screen1ViewBase
{
public:
    Screen1View();
    virtual ~Screen1View() {}
    virtual void setupScreen();
    virtual void tearDownScreen();

    virtual void toggleDifficulty();
    void handleTickEvent();
    virtual void getTick();

protected:
    bool difficulty;
    int randomTick;

};

#endif // SCREEN1VIEW_HPP
```

Screen1View.cpp:

```
#include <gui/screen1_screen/Screen1View.hpp>

Screen1View::Screen1View()
{
}

void Screen1View::setupScreen()
{
    Screen1ViewBase::setupScreen();
    difficultyButton.forceState(difficulty);
}

void Screen1View::tearDownScreen()
{
    Screen1ViewBase::tearDownScreen();
}

void Screen1View::toggleDifficulty()
{
    difficulty = difficultyButton.getState();
    presenter->storeDifficulty(difficulty);
}

void Screen1View::handleTickEvent()
{
    randomTick++;
}

void Screen1View::getTick()
{
    randomTick++;
    randomTick = randomTick * 1103515245 + 12345;
    randomTick = (randomTick / 65536) % 32768;
    presenter->storeTick(randomTick);
}
```

Screen2View.hpp:

```
#ifndef SCREEN2VIEW_HPP
#define SCREEN2VIEW_HPP

#include <gui_generated/screen2_screen/Screen2ViewBase.hpp>
#include <gui/screen2_screen/Screen2Presenter.hpp>

class Screen2View : public Screen2ViewBase
{
public:
    Screen2View();
    virtual ~Screen2View() {}
    virtual void setupScreen();
    virtual void tearDownScreen();

    virtual void handleTickEvent();
    void targetClicked();

    void setDifficulty(bool val);
    void setTick(int val);
    void reset();
protected:

    int tickCounter;
    int timePassed;

    int randomTime;
    int randomX;
    int randomY;
    int infinitick = 1;

    bool difficulty = false;
    bool allowSpawn = true;

};

#endif // SCREEN2VIEW_HPP
```

Screen2View.cpp:

```
#include <gui/screen2_screen/Screen2View.hpp>
#include <images/BitmapDatabase.hpp>

Screen2View::Screen2View()
{
}

void Screen2View::setDifficulty(bool val)
{
    difficulty = val;

    if (difficulty)
    {
        target.setBitmaps(Bitmap(BITMAP_TARGET20_MIN_ID),
Bitmap(BITMAP_TARGET20_MIN_ID));
    }
    else
    {
        target.setBitmaps(Bitmap(BITMAP_TARGET50_MIN_ID),
Bitmap(BITMAP_TARGET50_MIN_ID));
    }
    target.invalidate();
}

void Screen2View::setTick(int val)
{
    infinitick = val;
}

static uint16_t randomish(uint32_t seed)
{
    seed = seed * 1103515245 + 12345;
    return (seed / 65536) % 32768;
}

void Screen2View::setupScreen()
{
    Screen2ViewBase::setupScreen();
}
```

```

        tickCounter=0;
        timePassed = 0;
        randomTime = 180 + (randomish(infinitick) % 120);
    }

void Screen2View::tearDownScreen()
{
    Screen2ViewBase::tearDownScreen();
}

void Screen2View::handleTickEvent()
{
    tickCounter++;
    infinitick++;

    if(tickCounter == randomTime){

        infinitick++;
        randomX = 50 + (randomish(infinitick) % 380);
        infinitick++;
        randomY = 50 + (randomish(infinitick) % 170);
        target.setXY(randomX, randomY);
    }

    if(tickCounter >= randomTime && allowSpawn){

        target.setVisible(true);
        //target.invalidate();
        timePassed++;
    }
    target.invalidate();
}

void Screen2View::targetClicked()
{
    tickCounter = 0;
    double convertedTime = timePassed * 16.6667;

```

```
        Unicode::snprintf(textArealBuffer, TEXTAREAL_SIZE, "%d",
static_cast<int>(convertedTime));
        textAreal.invalidate();
        textAreal.setVisible(true);

        allowSpawn = false;

        timePassed = 0;

        target.setVisible(false);
        restart.setVisible(true);
        home.setVisible(true);

        restart.invalidate();
        home.invalidate();

        infinitick++;
        randomX = 50 + (randomish(infinitick) % 380);
        infinitick++;
        randomY = 50 + (randomish(infinitick) % 170);
        target.setXY(randomX, randomY);
    }

void Screen2View::reset()

{
    tickCounter = 0;
    timePassed = 0;
    textAreal.setVisible(false);

    allowSpawn = true;
    restart.setVisible(false);
    home.setVisible(false);
    restart.invalidate();
    textAreal.invalidate();
    home.invalidate();
}
```

Screen3View.hpp:

```
#ifndef SCREEN3VIEW_HPP
#define SCREEN3VIEW_HPP

#include <gui_generated/screen3_screen/Screen3ViewBase.hpp>
#include <gui/screen3_screen/Screen3Presenter.hpp>
#include <touchgfx/EasingEquations.hpp>
#include <touchgfx/mixins/FadeAnimator.hpp>

class Screen3View : public Screen3ViewBase
{
public:
    Screen3View();
    virtual ~Screen3View() {}
    virtual void setupScreen();
    virtual void tearDownScreen();
    virtual void handleTickEvent();
    void targetClicked();
    void setDifficulty(bool val);
    void setTick(int val);
    void reset();
    void endGame();

protected:
    //touchgfx::FadeAnimator< touchgfx::TextAreaWithOneWildcard > fade;
    int tickCounter;
    int timePassed;
    int randomTime;
    int randomX;
    int randomY;
    int numTargets = 0;
    int time = 30;
    int allTime = 0;
    int infinitick;
    bool difficulty = false;
    bool allowSpawn = true;
};

#endif // SCREEN3VIEW_HPP
```

Screen3View.cpp:

```
#include <gui/screen3_screen/Screen3View.hpp>
#include <images/BitmapDatabase.hpp>

Screen3View::Screen3View()
{
}

void Screen3View::setDifficulty(bool val)
{
    difficulty = val;
    if (difficulty){
        target.setBitmaps(Bitmap(BITMAP_TARGET20_MIN_ID),
Bitmap(BITMAP_TARGET20_MIN_ID));}
    else
    {
        target.setBitmaps(Bitmap(BITMAP_TARGET50_MIN_ID),
Bitmap(BITMAP_TARGET50_MIN_ID));
    }
    target.invalidate();
}

void Screen3View::setTick(int val)
{
    infinitick = val;
}

static uint16_t randomish(uint32_t seed)
{
    seed = seed * 1103515245 + 12345;
    return (seed / 65536) % 32768;
}

void Screen3View::setupScreen()
{
    Screen3ViewBase::setupScreen();
    allTime = 0;
    numTargets = 0;
    tickCounter=0;
    infinitick++;
    randomTime = 180 + (randomish(infinitick) % 120);
}
```

```

        infinitick++;
        randomX = 50 + (randomish(infinitick) % 380);
        infinitick++;
        randomY = 50 + (randomish(infinitick) % 170);
        target.setXY(randomX, randomY);
    }
}

void Screen3View::tearDownScreen()
{
    Screen3ViewBase::tearDownScreen();
}

void Screen3View::handleTickEvent()
{
    tickCounter++;
    if(tickCounter >= randomTime && allowSpawn){

        target.setVisible(true);
        target.invalidate();
        timePassed++;

        if(tickCounter % 60 == 0){
            time--;
            Unicode::snprintf(timerBuffer, TIMER_SIZE, "%d", time);
            timer.invalidate();
        }
    }
    target.invalidate();
    if(time <= 0){
        time = 0;
        endGame();
    }
}

void Screen3View::targetClicked()
{
    //tickCounter = 0;
    double convertedTime = timePassed * 16.6667;
    Unicode::snprintf(fadeBuffer, FADE_SIZE, "%d",
static_cast<int>(convertedTime));
    //fade.invalidate();
    fade.setPosition(198, 220, 84, 22);

```

```

        fade.setVisible(true);

        numTargets++;
        allTime += static_cast<int>(convertedTime);
        //allowSpawn = false;
        timePassed = 0;
        target.setVisible(false);
        //restart.setVisible(true);
        //home.setVisible(true);
        restart.invalidate();
        home.invalidate();
        fade.invalidate();
        infinitick++;
        randomX = 50 + (randomish(infinitick) % 380);
        infinitick++;
        randomY = 50 + (randomish(infinitick) % 170);
        target.setXY(randomX, randomY);
    }

void Screen3View::reset()
{
    tickCounter = 0;
    timePassed = 0;
    allTime = 0;
    numTargets = 0;
    time=30;
    Unicode::snprintf(timerBuffer, TIMER_SIZE, "%d", time);
    fade.setVisible(false);
    textAreal.setVisible(false);
    allowSpawn = true;
    restart.setVisible(false);
    home.setVisible(false);
    timer.setVisible(true);
    timer.invalidate();
    restart.invalidate();
    fade.invalidate();
    home.invalidate();
    textAreal.invalidate();
}

```

```

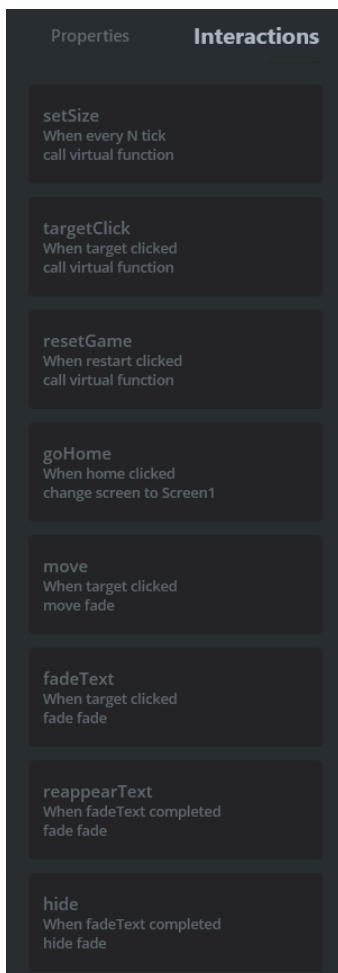
void Screen3View::endGame()
{
    tickCounter = 0;
    timePassed = 0;
    restart.setVisible(true);
    home.setVisible(true);
    target.setVisible(false);
    allowSpawn = false;
    Unicode::snprintf(textArea1Buffer1, TEXTAREA1BUFFER1_SIZE, "%d",
numTargets);
    Unicode::snprintf(textArea1Buffer2, TEXTAREA1BUFFER2_SIZE, "%d",
static_cast<int>(allTime/numTargets));
    textArea1.setVisible(true);
    timer.setVisible(false);

    target.invalidate();
    restart.invalidate();
    textArea1.invalidate();
    home.invalidate();
    timer.invalidate();
}

```

Vsakič, ko želimo vsiliti osveževanje elementa moramo poklicati funkcijo **invalidate()**.

V aplikaciji lahko ustvarimo interakcije. Interakcija se v našem primeru izvede, ko pritisnemo določen gumb kot je na primer tarča. Ob pritisku na gumb se pokliče funkcija TargetClicked().



Na sliki so prikazane različne interakcije, uporabljene na tretjem zaslonu. Na tem zaslonu je besedilo, ki prikazuje reakcijski čas za vsako tarčo. Prikaže se ob kliku na tarčo, interakcije pa omogočajo, da se premakne navzdol in v 0.3 s postane prosojno.

Naključne tarče:

Tarče se prikazujejo naključno, zato sem napisal kodo Randomish(), ki izračuna naključno vrednost vsakič, ko se pokliče:

```
static uint16_t randomish(uint32_t seed)
{
    seed = seed * 1103515245 + 12345;
    return (seed / 65536) % 32768;
}
```

Težava je v tem, da kot spremenljivko vedno sprejme vrednost, ki je vnaprej definirana. Če nastavimo vrednost `tickCounter` in je ta na začetku 0, bo vedno ob enakem času imela isto vrednost od začetka zaslona, kar pomeni, da bo sprva tarča vedno na enakem mestu.

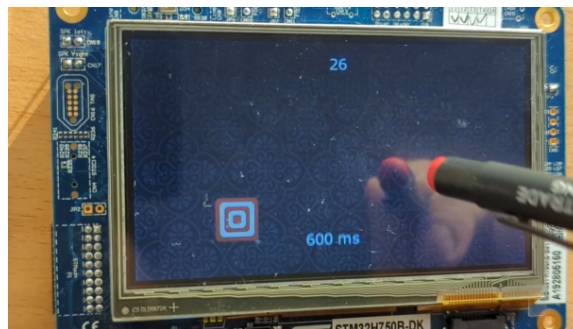
To sem popravil tako, da sem v `model.hpp` in `model.cpp` podal spremenljivko, v katero se shrani tick (ura) s prvega zaslona, le da ga tokrat ne shrani program, vendar uporabnik ko klikne na gumb za začetek igre 1 ali začetek igre 2. Uporabnik bo skoraj vedno kliknil v naključnem času.

`model.hpp`:

```
#ifndef MODEL_HPP
#define MODEL_HPP
class ModelListener;
class Model{
public:
    Model();
    void bind(ModelListener* listener)
    {
        modelListener = listener;
    }
    void tick();
    void storeDifficulty(bool val);
    bool getDifficulty() const;
    void storeTick(int val);
    int getTick() const;
protected:
    ModelListener* modelListener;
    bool difficulty = false;
    int randomTick = 1;
};
#endif // MODEL_HPP
```

model.cpp:

```
#include <gui/model/Model.hpp>
#include <gui/model/ModelListener.hpp>
Model::Model() : modelListener(0)
{
}
void Model::tick()
{
}
void Model::storeDifficulty(bool val)
{
    difficulty = val;
}
bool Model::getDifficulty() const
{
    return difficulty;
}
void Model::storeTick(int val)
{
    randomTick = val;
}
int Model::getTick() const
{
    return randomTick;
}
```



Video demonstracija: <https://youtu.be/ESDqSOgw8tY>

Kako zaženemo aplikacijo: V TouchGFX Designer odpremo datoteko ReactionTimer.touchgfx v mapi TouchGFX. Ko se projekt naloži kliknemo na gumb generate code in run & build. To bo naložilo še vse potrebne knjižnice.

Projekt je objavljen na GitHub strani:

https://github.com/mattbernot/ReactionTimer_STM32H750B

2. Neobvezna domača naloga:

Za 2. Neobvezno domačo nalogo sem se odločil, da naredim igro CookieClicker:



Video demonstracija: <https://youtu.be/qk-OavF2OSA>

Postopek izdelave je razložen v poročilu 2. neobvezne domače naloge.