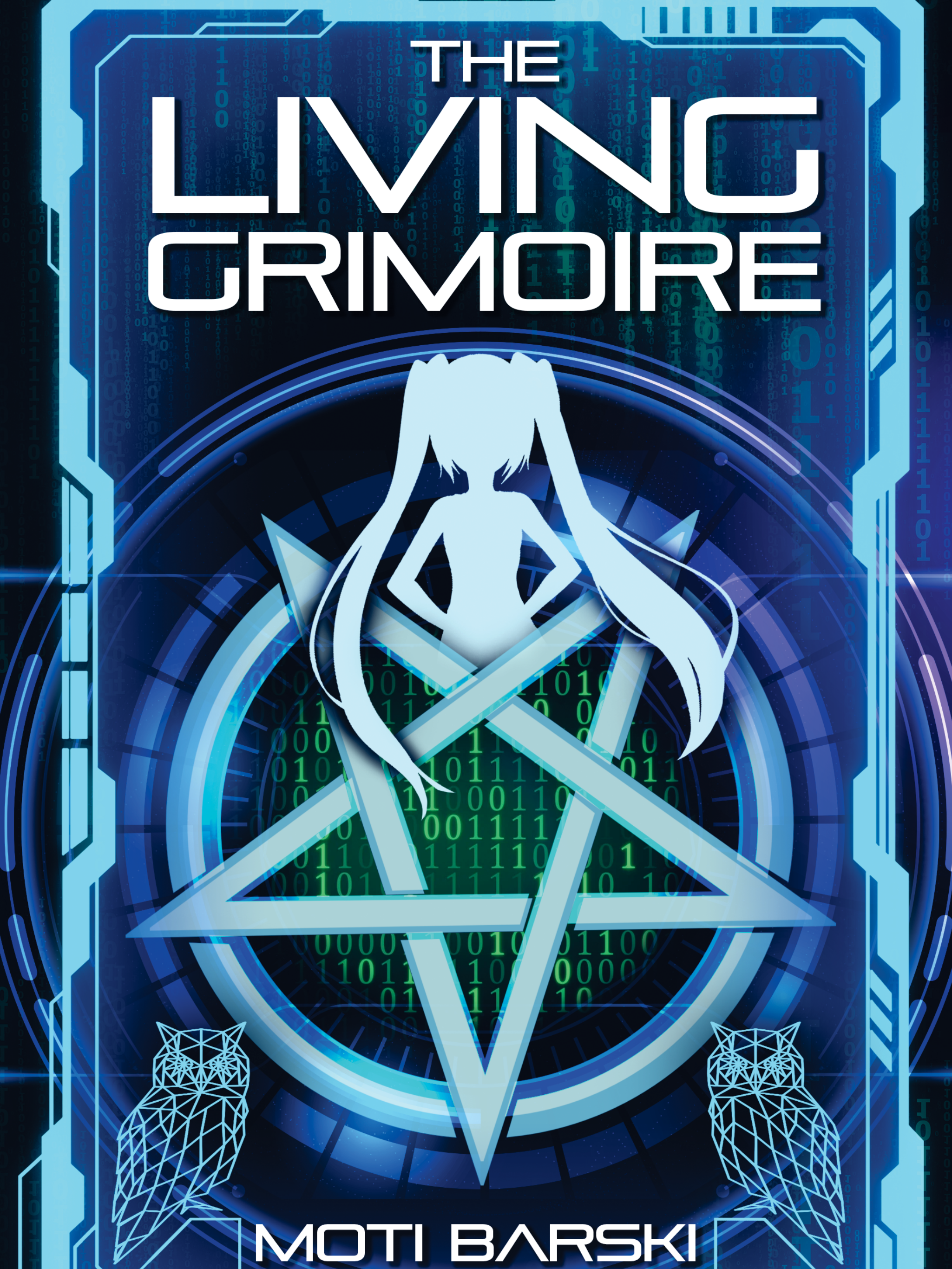
THE LIVING GRIMOIRE



MOTI BARSKI

written by Moti Barski.

I moti barski do not allow anyone and or anybody and or any organization to receive monetary profit from this living grimoire unless approved in writing by me personally. you can use this for research.

LivinGrimoire is a software design pattern that absorbs skills with just one line of code needed to add a skill.

Advantages of LivinGrimoire

1. Skill Prioritization: Can prioritize skills against one another, pause and resume skills according to their priority.

2. Algorithm Queueing: Can queue algorithms while running other algorithms.

3. Concurrent Skill Engagement: Can engage several skills at once and engage the right skills.

4. Inter-Skill Communication: Skills can communicate with each other, pass data, and affect each other.

5. Cross-Platform Compatibility: Interface is not used, making it applicable for all OOP programming languages.

6. Auxiliary Classes: Specialized for learnability and trigger management, with miscellaneous classes for time savings on common coding actions.

7. Multistep Algorithms: Can form multistep algorithms and abort algorithms while they run.

8. Built-In Skill Catalog: Features a built-in skill catalog (see wiki to learn more).

9. Dynamic Skill Management: Can use self-aware skills, enabling them to add or remove other skills while the program is running, similar to a command terminal (see TheShell wiki).

method of operation

the LG can absorb skills and use them.

1) a skill sends out Algorithms if triggered.

2) said Algorithm is built with a list of parts (Mutatable class).

3) as the algorithm is running the action method of each mutatable is engaged

outputting a string.

* a running algorithm can also be aborted at any time by setting the mutatable (AlgPart super class) algKillSwitch attribute to true.

see AP classes for Mutatable class examples

see DiHello world class for skill example

getting started

to use the living grimoire choose the programming language you prefer.

and paste the LivinGrimoire files.

to keep LG projects neat and tidy I recommend separating the project into

at least 2 packages:

1 **LGCore** : short for living grimoire core,

this are the core class files that compose the AGI software design pattern.

2 SkillsPkg : this directory should contain :

skill classes (naming convention : class names starts with Di)

AlgParts (naming convention : class names starts with AP)

hello world

DiHello world is an example skill that says hello world as a reply to hello :

```
public class DiHelloWorld extends Skill {
    // hello world skill for testing purposes
    public DiHelloWorld() {
        super();
    }
    @Override
    public void input(String ear, String skin, String eye) {
        switch (ear){
            case "hello":
                super.setSimpleAlg("hello world");
                break;
        }
    }
    @Override
    public String skillNotes(String param) {
        if ("notes".equals(param)) {
            return "plain hello world skill";
        } else if ("triggers".equals(param)) {
            return "say hello";
        }
        return "note unavailable";
    }
}
```

example use in main:

```
Chobits chi = new Chobits();
chi.addSkill(new DiHelloWorld());
System.out.println(chi.think("hello","",""));
System.out.println(chi.think("","",""));
```

output:

hello world

CHOBITS CLS: API

```
public class Chobits {
    public Chobits();
    public void setDataBase(AbsDictionaryDB absDictionaryDB);
    // your DB should override simple load and save functions.
    public Chobits addSkill(Skill skill);
    public Chobits addSkillAware(Skill skill);
    // these skills get the Chobit Object shallow ref as an attribute
    // see Artificial Intelligence hormones wiki for more
    public void clearSkills();
    public void addSkills(Skill... skills);
    public void removeSkill(Skill skill);
    public Boolean containsSkill(Skill skill);
    public String think(String ear, String skin, String eye);
    // this method needs to run in each think cycle. it engages the equipped skills.
    public String getSoulEmotion();
    // returns alg part's name (it represents an emotion)
    protected void inOut(Skill dClass, String ear, String skin, String eye);
    public Kokoro getKokoro();
    public void setKokoro(Kokoro kokoro);
    public Fusion getFusion();
    //(underuse) you may need this method if you want to override the think method.
    public ArrayList<String> getSkillList();
}
```

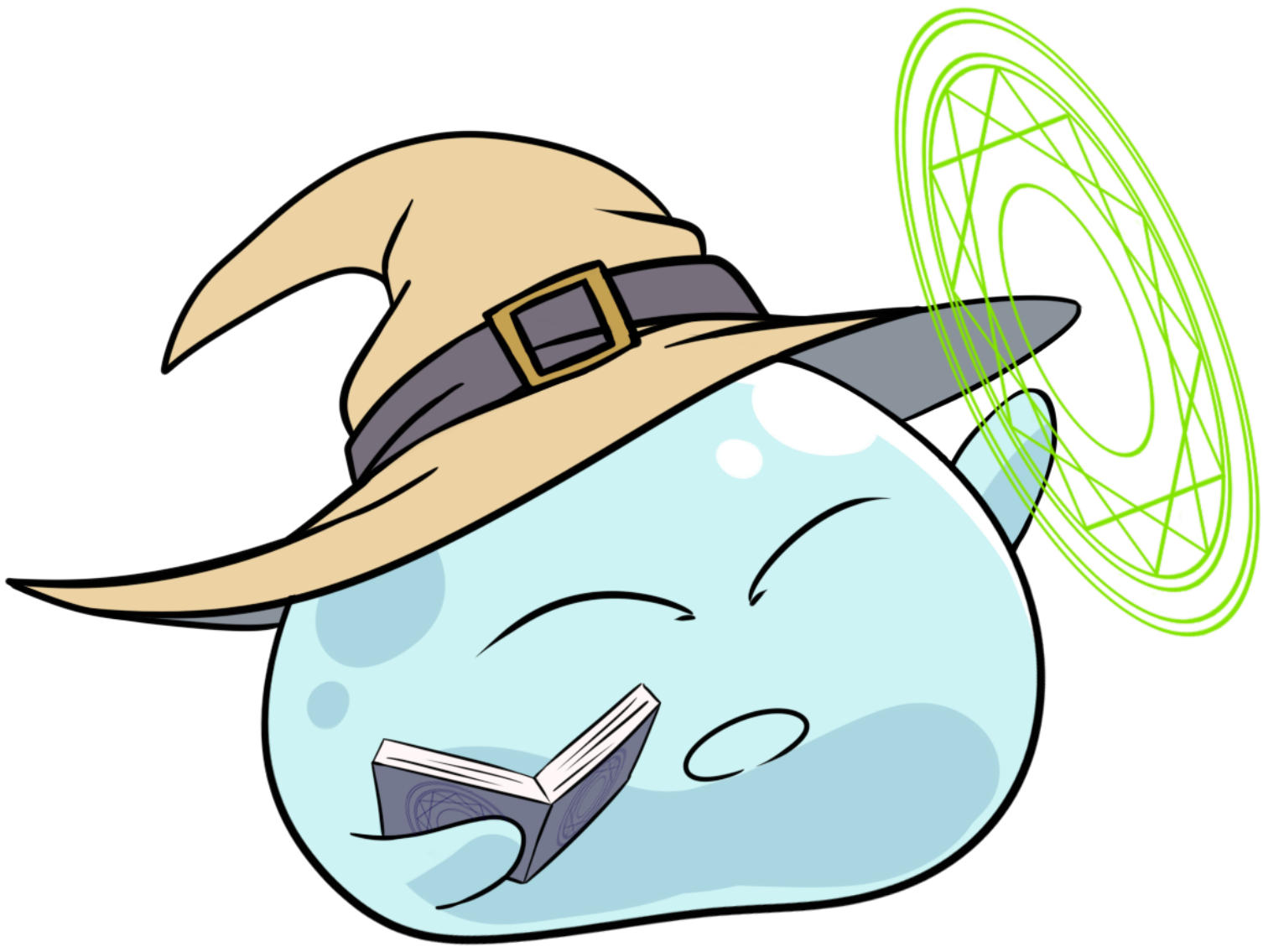
SKILL CLS: API

each Skill overrides this class.

```
public class Skill {
    protected Kokoro kokoro = null;
    protected Algorithm outAlg = null;
    protected int outpAlgPriority = -1;

    public Skill();
    public void input(String ear, String skin, String eye);
    // skill's triggers, logic, and output goes here.
    protected void setVerbatimAlg(int priority, String... sayThis);
// build an algorithm to output string_N per think cycle.
// set algorithm priority 1->5 with 5 being the lowest priority to run.
protected void setSimpleAlg(String... sayThis);
// build an algorithm to output string_N per think cycle.
// set algorithm priority to a default of 4.
protected void setVerbatimAlgFromList(int priority, ArrayList<String> sayThis);
protected void algPartsFusion(int priority, Mutatable... algParts);
// use custom non defauld alg Parts(Mutatable sub classes) in your algorithm
// override the action logic. the class's name can represents emotion.
public String strContainsList(String str1, ArrayList<String> items);
// does str1 contain an item from items? this method can be used for some triggers
// but it depends on the trigger logic you want to implement.
```

```
    // I like having it accessible.
    public String skillNotes(String param);
    // see "built in skill catalog" wiki
}
```



KOKORO CLS API

use this attribute in your skills if needed to communicate between skills.

```
public class Kokoro {
    private String emot = "";

    public String getEmot();
    // getter of string
    public void setEmot(String emot);
    // setter of a string
    public GrimoireMemento grimoireMemento;
    // access Chobit's database(it has simple save and load methods)
    public Hashtable<String, String> toHeart;
    // use this dictionary to communicate between skills
    public Kokoro(AbsDictionaryDB absDictionaryDB);
    // set the database object(if needed)
}
```

BRAIN CLS: API

the Brain class is the amalgamation of 2 chobits.

one for regular(logical) skills, and one for hardware.

the output of the 1st goes in the input of the 2nd.

this way the proper hardware(TextToSpeech, GUI changes, think cycle speed) can be engaged.

tho hardware skills can also be engaged as logical skills, absorbing

hardware skills into the hardware Chobit object of the Brain object

offer arbitration.

note the Kokoro object is shared for both chobits.

```
public class Brain {
    public Chobits logicChobit;
    public Chobits hardwareChobit;
    public String getEmotion();
    // get logical chobits active alg part's name.
    public String getLogicChobitOutput();
    public Brain();
    public void doIt(String ear, String skin, String eye);
    // this method must be engaged in each (think) cycle.
    public void addLogicalSkill(Skill skill);
    // add a skill to the logical chobit
    public void addHardwareSkill(Skill skill);
    // add a skill to the hardware chobit
}
```

suggested naming convention

AP

AlgParts class names should start with AP. For example, APSay.

Di

Skill (LivinGrimoire skill class names) should start with Di. For example, DiHelloWorld.

Da

DaSkills are skills that are asynchronous.

AX

Auxiliary modules. These classes names can start with AX. For example, AXLearnability.

AH

AI Hormone skills. These are skills with Chobits in their constructor, which enables them to add or remove other skills the Chobit object is equipped with. Each AI Hormone skill has its own logic. For example, AHMoodRegulator.

links

video course links at https://github.com/yotamarker/public-livinGrimoire

wikis: https://github.com/yotamarker/public-livinGrimoire and click wikis

official forum: https://www.yotamarker.com/

A DETAILED ELABORATION OF THE MOST EFFICIENT WAY TO PROGRAM.
LEARN ABOUT THE ARTIFICIAL GENERAL INTELLIGENCE SOFTWARE DESIGN PATTERN CALLED THE LIVING GRIMOIRE, WHICH AMONG ITS MANY CAPABILITIES, ENABLES ADDING SKILLS USING ONLY ONE LINE OF CODE PER SKILL.