```
/
*----------------------------------------
-----------*
|          Probability  Theory          |
|             by ducdat0507             |
|            "i give up lmao"           |
|         v.5:  the perk update         |
*----------------------------------------
-----*/

import { ExponentialCost, LinearCost,
ConstantCost, CompositeCost,
FreeCost, StepwiseCost } from "./api/
Costs";
import { Localization } from "./api/
Localization";
import { BigNumber } from "./api/
BigNumber";
import { theory } from "./api/Theory";
import { Utils } from "./api/Utils";
import { TextAlignment } from "./api/ui/
properties/TextAlignment";
import { Thickness } from "./api/ui/
properties/Thickness";
import { Color } from "./api/ui/
properties/Color";
import { LayoutOptions } from "./api/ui/
properties/LayoutOptions";
import { TouchType } from "./api/ui/
properties/TouchType";
import { Currency } from "./api/
Currency";
import { Easing } from "./api/ui/
properties/Easing";
import { FontFamily } from "./api/ui/
properties/FontFamily";
import { ImageSource } from "./api/ui/
properties/ImageSource";
import { StackOrientation } from "./api/
ui/properties/StackOrientation";
```

```
var id = "probability_theory";
var name = "Probability Theory";
var getDescription = () => {

    let day = Math.floor(Date.now() / 86400000);
    let seed = day * 9999991;

    let descRoll = [
        "A theory about probabilities and loot boxes",
        "A theory about gacha, loot boxes, and everything random",
        "A theory about researching the randomness",
        "A theory about random stuff",
    ];
    let subDescRoll = [
        "a randomized description generator",
        "some achievements",
        "minigames",
        "multiple game stages",
        "literally a gachapon system",
        "some die rolling",
    ];

    let desc = descRoll[seed % descRoll.length] + ". Featuring ";
    seed = Math.floor((seed * seed / 1000) % 9999991);

    for (let a = 0; a < 2; a++) {
        let t = Math.floor(seed % subDescRoll.length);
        desc += subDescRoll[t] + ", ";
        seed = Math.floor((seed * seed / 1000) % 9999991);
        subDescRoll.splice(t, 1);
    }
    desc += "and " + subDescRoll[Math.floor(seed % subDescRoll.length)] + ".";
    seed = Math.floor((seed * seed / 1000) % 9999991);

    let tips = [
        "The description of this theory will change every day! Or if I decide to update the description generator.",
        "This theory is getting less and less about the maths behind probability and more and more about random things I could think about.",
        "This theory is compatible with your ad rewards. It isn't compatible with the minigames though, so I decided to make my own ones.",
        "EX skills appear half as often as normal skills, while SP skills, when unlocked, are 50 times rarer than normal skills.",
        "If your tau becomes larger than 1e10000, you can still use the custom theory selection menu to view the full exponent number",
    ];

    desc += "\n\nTip of the day: " + tips[seed % tips.length];

    desc += "\n\nVersion: 5, Patch: 9, Mod: 1"

    return desc;
}
var authors = "ducdat0507 (former), \nAlex/hax";
var version = 4;
var currency, currency2, currency3;
var stage = 0;
var prg = BigNumber.ZERO, prg2 =
BigNumber.ZERO, prgGacha =
BigNumber.ZERO, prg3 =
BigNumber.ZERO, prgDice =
BigNumber.ZERO;
var prgBar;
var r = [BigNumber.ONE,
BigNumber.ONE, BigNumber.ONE,
BigNumber.ONE,
    BigNumber.ZERO, BigNumber.ZERO,
BigNumber.ZERO, BigNumber.ZERO];
var q = [BigNumber.ONE,
BigNumber.ONE, BigNumber.ONE,
BigNumber.ONE,
    BigNumber.ZERO, BigNumber.ZERO,
BigNumber.ZERO, BigNumber.ZERO];

var stars = [0, 0, 0, 0, 0, 0];

var gacha = BigNumber.ZERO,
gachaTotal = BigNumber.ZERO;
var dicePoints = BigNumber.ZERO,
dicePoints2 = BigNumber.ZERO;
var b1, b2, b3;
var c1, c2, c3, c4, c5, c6, c7, c8, c9,
c10, c11, c12;
var d4, d6, d8, d10, d12, d20;
var n1, n2, n3, n4;
var clicker, minigame, perk, booster;

var permPub, permAll, permAuto;
var perm1, perm2, perm3, perm4;
var perm5, perm6, perm7, perm8;
var perm9, perm10, perm11, perm12;
var perm13, perm14, perm15;
var d4Master, d6Master, d8Master,
d10Master, d12Master, d20Master;
var diceMasteries =
"....................................";
```

```
var skills = [], skillExp = [];

var ac1, ac2, ac3, ac4, ac5, ac6;

var sc1Prog = 0;

var prgSpeed, rTerms, cTerms,
qTerms, cur2Unlock, gachaUnlock,
gachaValue, gachaBulk, cur3Unlock,
diceUnlock;
var prg3Speed, qTerms3,
gachaValue2, rpgUnlock, workNerf,
atkBoost, diceBoost, diceBoost2;
var quaternaryEntries = [];

var rpgTheory = 1, rpgLemma = 1,
rpgBestLemma = 0, rpgScore = 0;
var ourHealth = BigNumber.ONE,
theirHealth = BigNumber.ONE,
theirMaxHealth = BigNumber.ZERO;

var hiscores = [0, 0, 0];
var hiscoreEffs = [BigNumber.ONE,
BigNumber.ONE, BigNumber.ONE];

var skillData = [
    {
        name: (amount) => "Accelerator $
\\rho_1$",
        info: (amount) =>
Localization.getUpgradeMultCustomIn
fo("\\rho_1\\text{ speed}", "{eff}"),
        effect: (level) => 1 + level * .1,
        effectText: (level) => "+0.1 /
level",
        starValue: [1, 3, 7, 17, 37, 77],
        starCost: new LinearCost(1, 1),
    },
    {
        name: (amount) => "Accelerator $
```

```
\\rho_2$",
        info: (amount) =>
Localization.getUpgradeMultCustomIn
fo("\\rho_2\\text{ speed}", "{eff}"),
        effect: (level) => 1 + level * .1,
        effectText: (level) => "+0.1 /
level",
        starValue: [1, 3, 7, 17, 37, 77],
        starCost: new LinearCost(1, 1),
    },
    {
        name: (amount) => "Accelerator $
\\ominus$",
        info: (amount) =>
Localization.getUpgradeMultCustomIn
fo("\\ominus\\text{ speed}", "{eff}"),
        effect: (level) => 1 + level * .05,
        effectText: (level) => "+0.05 /
level",
        starValue: [1, 2, 4, 8, 16, 32],
        starCost: new LinearCost(0, 5),
    },
    {
        name: (amount) => "Booster $\
\rho_1$",
        info: (amount) =>
Localization.getUpgradeMultCustomIn
fo("\\rho_1\\text{ gain}", "{eff}"),
        maxLevel: 50000000,
        effect: (level) =>
BigNumber.TEN.pow(level),
        effectText: (level) => "$\
\times$1000  / level",
        starValue: [100, 300, 700, 1700,
3700, 7700],
        starCost: new LinearCost(0, 2),
    },
    {
        name: (amount) => "Booster $\
\rho_2$",
        info: (amount) =>
```

```
Localization.getUpgradeMultCustomIn
fo("\\rho_2\\text{ gain}", "{eff}"),
        maxLevel: 50000000000,
        effect: (level) =>
BigNumber.TWO.pow(level),
        effectText: (level) => "$\
\times$9999 / level",
        starValue: [100, 300, 700, 1700,
3700, 7700],
        starCost: new LinearCost(0, 2),
    },
    {
        name: (amount) => "Booster $\
\bar\\tau$",
        info: (amount) =>
Localization.getUpgradeMultCustomIn
fo("\\text{publication multiplier}",
"{eff}"),
        maxLevel: 5000000000000,
        effect: (level) =>
BigNumber.THREE.pow(level),
        effectText: (level) => "$\
\times$30000 / level",
        starValue: [100, 300, 700, 1700,
3700, 7700],
        starCost: new LinearCost(0, 2),
    },
    {
        name: (amount) => "Raiser
$b_1$",
        info: (amount) =>
Localization.getUpgradeIncCustomExp
Info("\\text{base } b_1", "{eff}"),
        maxLevel: 300000,
        effect: (level) => 1 + level * .05,
        effectText: (level) => "+0.05 /
level",
        starValue: [1, 3, 9, 27, 81,
243000],
        starCost: new LinearCost(5, 6),
    },
    {
        name: (amount) => "Raiser
$b_2$",
        info: (amount) =>
Localization.getUpgradeIncCustomExp
Info("\\text{base } b_2", "{eff}"),
        maxLevel: 30,
        effect: (level) => 1 + level * .05,
        effectText: (level) => "+0.05 /
level",
        starValue: [1, 3, 9, 27, 81,
2430000],
        starCost: new LinearCost(5, 6),
    },
    {
        name: (amount) => "\\textsf{EX}
Public Link $\\rho_2$",
        info: (amount) => "Publication
multiplier affects $\\rho_2$ at {eff}\\%
{} efficiency",
        maxLevel: 100000000,
        effect: (level) => level * .5,
        effectText: (level) => "+0.5\\%{} /
level",
        starValue: [100000, 2000000, 3,
5, 7, 11],
        starCost: new ExponentialCost(5,
1),
    },
    {
        name: (amount) => "\\textsf{EX}
Tachyon Maker",
        info: (amount) => "Reduces
$r_{5~8}$ decaying rate",
        effect: (level) => 20 + level,
        effectText: (level) => "Formula: $\
\dot{r_i} = \\frac{-r_i}{20 + \
\text{level}}$",
        starValue: [10000000, 2, 3, 5, 7,
11],
        starCost: new LinearCost(5, 1),
    },
    {
        name: (amount) => "\\textsf{EX}
Extra Supply $c_4$",
        info: (amount) =>
Localization.getUpgradeIncCustomInf
o("\\text{maximum }c_4\\text{ level}",
"{eff}"),
        effect: (level) => level * 2,
        effectText: (level) => "+2 / level",
        starValue: [10000000, 2, 3, 5, 7,
11],
        starCost: new LinearCost(5, 5),
    },
    {
        name: (amount) => "\\textsf{EX}
Variable Bundle",
        info: (amount) => "Unlocks
$c_{5~8}$, depending on level",
        maxLevel: 4,
        effect: (level) => level,
        effectText: (level) => "$c_" +
(level + 4) + "$ unlocked" + (level <
4 ? ", next unlocks $c_" + (level + 5) +
"$" : ""),
        starValue: [1, 2, 3, 5, 7, 11],
        starCost: new
ExponentialCost(20, 2),
    },
    {
        name: (amount) => "\\textsf{EX}
Puzzle Package",
        info: (amount) => "Unlocks new
minigames, depending on level",
        maxLevel: 2,
        effect: (level) => level,
        effectText: (level) => {
            let minigames = ["Block
Puzzle", "Domino Puzzle", "Tile
Puzzle"];
            return minigames[level - 1] + "
unlocked" + (level < 3 ? ", next unlocks
" + minigames[level] : "")
        },
        starValue: [1, 2, 3, 5, 7, 11],
        starCost: new
ExponentialCost(50, 2),
    },
    {
        name: (amount) => "\\textsf{SP}
Accelerator $\\rho_3$",
        info: (amount) =>
Localization.getUpgradeMultCustomIn
fo("\\rho_3\\text{ speed}", "{eff}") + ",
based on the first 3 generators'
levels",
        effect: (level) => 1 +
(skills[0].level + skills[1].level +
skills[2].level) * 0.0005 * level,
        effectText: (level) => "+" +
((skills[0].level + skills[1].level +
skills[2].level) * 0.0005).toFixed(3) +
" / level",
        starValue: [1, 2, 3, 4, 5, 6],
        starCost: new LinearCost(225,
25),
    },
    {
        name: (amount) => "\\textsf{SP}
Accelerator $\\boxdot$",
        info: (amount) =>
Localization.getUpgradeMultCustomIn
fo("\\boxdot\\text{ speed}", "{eff}") + ",
based on the first 3 generators'
levels",
        effect: (level) => 1 +
(skills[0].level + skills[1].level +
skills[2].level) * 0.0002 * level,
        effectText: (level) => "+" +
((skills[0].level + skills[1].level +
skills[2].level) * 0.0002).toFixed(3) +
" / level",
        starValue: [1, 2, 3, 4, 5, 6],
        starCost: new LinearCost(225,
25),
    },
    {
        name: (amount) => "\\textsf{SP}
Booster $\\rho_3$",
        info: (amount) =>
Localization.getUpgradeMultCustomIn
fo("\\rho_3\\text{ gain}", "{eff}") + ",
based on the first 3 boosters' levels",
        maxLevel: 30,
        effect: (level) =>
BigNumber.from(1 + (skills[3].level +
skills[4].level + skills[5].level) *
0.0005).pow(level),
        effectText: (level) => "$\\times$"
+ (1 + (skills[3].level + skills[4].level +
skills[5].level) * 0.0005).toFixed(3) +
" / level",
        starValue: [1, 2, 3, 4, 5, 6],
        starCost: new LinearCost(250,
50),
    },
    {
        name: (amount) => "\\textsf{SP}
Raiser $b_3$",
        info: (amount) =>
Localization.getUpgradeIncCustomExp
Info("\\text{base } b_3", "{eff}") + ",
based on the first 2 raisers' levels",
        maxLevel: 15,
        effect: (level) => 1 +
(skills[6].level + skills[7].level) *
0.0005 * level,
        effectText: (level) => "+" +
((skills[6].level + skills[7].level) *
0.0005).toFixed(3) + " / level",
        starValue: [1, 2, 3, 4, 5, 6],
        starCost: new LinearCost(225,
75),
    },
    {
        name: (amount) => "\\textsf{SP}
Amplifier $\\boxdot \\rightarrow \
\ominus$",
        info: (amount) => "Improves $\
\boxdot$ effect to $\\ominus$",
        maxLevel: 20,
        effect: (level) => 3 - level * 0.05,
        effectText: (level) => "Formula: $\
\sqrt[(3 - 0.05 \\times \\text{level})]{\
\log(\\boxdot + 1) + 1}$",
        starValue: [1, 2, 3, 4, 5, 6],
        starCost: new LinearCost(200,
100),
    },
    {
        name: (amount) => "\\textsf{SP}
Amplifier $\\boxdot \\rightarrow \
\rho_3$",
        info: (amount) => "Improves $\
\boxdot$ effect to $\\rho_3$",
        maxLevel: 30,
        effect: (level) => 2 - level * 0.05,
        effectText: (level) => "Formula: $\
\sqrt[(2 - 0.05 \\times \\text{level})]{\
\boxdot + 1}$",
        starValue: [1, 2, 3, 4, 5, 6],
        starCost: new LinearCost(300,
100),
    },
    {
        name: (amount) => "\\textsf{SP}
Public Link $\\rho_3$",
        info: (amount) => "Publication
multiplier affects $\\rho_3$ at {eff}\\%
{} efficiency",
        maxLevel: 20,
        effect: (level) => level * .05,
        effectText: (level) => "+0.05\\%
{} / level",
        starValue: [1, 2, 3, 4, 5, 6],
        starCost: new StepwiseCost(new
ExponentialCost(400, 1), 3),
    },
    {
        name: (amount) => "\\textsf{SP}
Variable Bundle$^2$",
        info: (amount) => "Unlocks
$c_{9~12}$, depending on level",
        maxLevel: 4,
        effect: (level) => level,
        effectText: (level) => "$c_" +
(level + 8) + "$ unlocked" + (level <
4 ? ", next unlocks $c_" + (level + 9) +
"$" : ""),
        starValue: [1, 2, 3, 4, 5, 6],
        starCost: new
ExponentialCost(200, 2),
    },
]

var getSkillEffect = (id) =>
skillData[id].effect(skills[id].level);

var perks = { }
var perkPoints = 0
var perkPointsBought = 0

var pBoosters = [0, 0, 0, 0];
var pBoostersBought = 0;

var perkData = {
    11: {
        name: "Dice Factions",
        info: "Unlocks types for theory
lemma and \\textsf{ATK} boosters that
are based on them.",
    },
    21: {
        name: "Libraries",
        info:
```

```
Localization.getUpgradeMultCustomIn
fo("$\\textsf{ATK} to Data Collection
lemmas$", "2"),
        req: [11],
    },
    22: {
        name: "Calculators",
        info:
Localization.getUpgradeMultCustomIn
fo("$\\textsf{ATK} to Analyzation
lemmas$", "2"),
        req: [11],
    },
    23: {
        name: "Data Check Algorithm",
        info:
Localization.getUpgradeMultCustomIn
fo("$\\textsf{ATK} to Evaluation
lemmas$", "2"),
        req: [11],
    },
    24: {
        name: "Co-researchers",
        info:
Localization.getUpgradeMultCustomIn
fo("$\\textsf{ATK} to Revision
lemmas$", "2"),
        req: [11],
    },
    31: {
        name: "The Internet",
        info:
Localization.getUpgradeIncCustomInf
o("$\\textsf{ATK} multiplier per Data
Collection upgrade$", "1"),
        req: [21],
    },
    32: {
        name: "Computers",
        info:
Localization.getUpgradeIncCustomInf
```

```
o("$\\textsf{ATK} multiplier per
Analyzation upgrade$", "1"),
        req: [22],
    },
    33: {
        name: "Citations",
        info:
Localization.getUpgradeIncCustomInf
o("$\\textsf{ATK} multiplier per
Evaluation upgrade$", "1"),
        req: [23],
    },
    34: {
        name: "Co-PHDs",
        info:
Localization.getUpgradeIncCustomInf
o("$\\textsf{ATK} multiplier per
Revision upgrade$", "1"),
        req: [24],
    },
    41: {
        name: "Experience",
        info:
Localization.getUpgradeMultCustomIn
fo("$\\textsf{ATK}$", "\\frac{\\text{best
theory}}{5} + 1"),
        req: [31, 32],
    },
    42: {
        name: "Short Term Memory",
        info:
Localization.getUpgradeMultCustomIn
fo("$\\textsf{ATK}$", "\\frac{\\text{best
lemma}}{25} + 1"),
        req: [33, 34],
    },
};

var init = () => {
    currency = theory.createCurrency();
    currency2 =
theory.createCurrency();
    currency3 =
theory.createCurrency();

    //////////////////
    // Regular Upgrades


    // v1

    // b1
    {
        let getDesc = (level) => "b_1=" +
(getB1(level) / (r[7] + 1)).toString(0) +
(perm4.level > 0 ? "\\times (r_8 + 1)" :
"");
        let getInfo = (level) => "b_1=" +
getB1(level).toString();
        b1 = theory.createUpgrade(0,
currency, new ExponentialCost(25,
Math.log2(1.5)));
        b1.getDescription = (_) =>
Utils.getMath(getDesc(b1.level));
        b1.getInfo = (amount) =>
Utils.getMathTo(getInfo(b1.level),
getInfo(b1.level + amount));
    }
    // c1
    {
        let getDesc = (level) => "c_1=" +
(getC1(level) / (r[6] + 1)).toString(0) +
(perm3.level > 0 ? "\\times (r_7 + 1)" :
"");
        let getInfo = (level) => "c_1=" +
getC1(level).toString();
        c1 = theory.createUpgrade(1,
currency, new ExponentialCost(100,
Math.log2(2)));
        c1.getDescription = (_) =>
Utils.getMath(getDesc(c1.level));
        c1.getInfo = (amount) =>
```

```
Utils.getMathTo(getInfo(c1.level),
getInfo(c1.level + amount));
    }
    // c2
    {
        let getDesc = (level) =>
"c_2=2^{" + level + "}";
        let getInfo = (level) => "c_2=" +
getC2(level).toString(0);
        c2 = theory.createUpgrade(2,
currency, new ExponentialCost(1000,
Math.log2(20)));
        c2.getDescription = (_) =>
Utils.getMath(getDesc(c2.level));
        c2.getInfo = (amount) =>
Utils.getMathTo(getInfo(c2.level),
getInfo(c2.level + amount));
    }
    // c3
    {
        let getDesc = (level) => "c_3=" +
getC3(level).toString(0);
        c3 = theory.createUpgrade(3,
currency, new ExponentialCost(1e80,
Math.log2(120)));
        c3.getDescription = (_) =>
Utils.getMath(getDesc(c3.level));
        c3.getInfo = (amount) =>
Utils.getMathTo(getDesc(c3.level),
getDesc(c3.level + amount));
    }
    // c4
    {
        let getDesc = (level) =>
"c_4=2^{" + level + "}";
        let getInfo = (level) => "c_4=" +
getC4(level).toString(0);
        c4 = theory.createUpgrade(4,
currency, new ExponentialCost(1e120,
32));
        c4.maxLevel = 16;
        c4.getDescription = (_) =>
Utils.getMath(getDesc(c4.level));
        c4.getInfo = (amount) =>
Utils.getMathTo(getInfo(c4.level),
getInfo(c4.level + amount));
    }

    // b2
    {
        let getDesc = (level) => "b_2=" +
getB2(level).toString();
        b2 = theory.createUpgrade(10,
currency2, new ExponentialCost(25,
Math.log2(2)));
        b2.getDescription = (_) =>
Utils.getMath(getDesc(b2.level));
        b2.getInfo = (amount) =>
Utils.getMathTo(getDesc(b2.level),
getDesc(b2.level + amount));
    }
    // n1
    {
        let getDesc = (level) => "n_1=" +
getN1(level).toString(0);
        n1 = theory.createUpgrade(11,
currency2, new ExponentialCost(100,
Math.log2(2)));
        n1.getDescription = (_) =>
Utils.getMath(getDesc(n1.level));
        n1.getInfo = (amount) =>
Utils.getMathTo(getDesc(n1.level),
getDesc(n1.level + amount));
    }
    // n2
    {
        let getDesc = (level) =>
"n_2=2^{" + level + "}";
        let getInfo = (level) => "n_2=" +
getN2(level).toString(0);
        n2 = theory.createUpgrade(12,
currency2, new ExponentialCost(1000,
Math.log2(1000)));
        n2.getDescription = (_) =>
Utils.getMath(getDesc(n2.level));
        n2.getInfo = (amount) =>
Utils.getMathTo(getInfo(n2.level),
getInfo(n2.level + amount));
    }
    // c5
    {
        let getDesc = (level) =>
"c_5=2^{" + level + "}";
        let getInfo = (level) => "c_5=" +
getC5(level).toString(0);
        c5 = theory.createUpgrade(13,
currency2, new ExponentialCost(1e12,
Math.log2(1e6)));
        c5.getDescription = (_) =>
Utils.getMath(getDesc(c5.level));
        c5.getInfo = (amount) =>
Utils.getMathTo(getInfo(c5.level),
getInfo(c5.level + amount));
        c5.isAvailable = false;
    }
    // c6
    {
        let getDesc = (level) =>
"c_6=3^{" + level + "}";
        let getInfo = (level) => "c_6=" +
getC6(level).toString(0);
        c6 = theory.createUpgrade(14,
currency2, new ExponentialCost(1e16,
Math.log2(1e12)));
        c6.getDescription = (_) =>
Utils.getMath(getDesc(c6.level));
        c6.getInfo = (amount) =>
Utils.getMathTo(getInfo(c6.level),
getInfo(c6.level + amount));
        c6.isAvailable = false;
    }
    // c7
    {
        let getDesc = (level) =>
"c_7=5^{" + level + "}";
        let getInfo = (level) => "c_7=" +
getC7(level).toString(0);
        c7 = theory.createUpgrade(15,
currency2, new ExponentialCost(1e32,
Math.log2(1e24)));
        c7.getDescription = (_) =>
Utils.getMath(getDesc(c7.level));
        c7.getInfo = (amount) =>
Utils.getMathTo(getInfo(c7.level),
getInfo(c7.level + amount));
        c7.isAvailable = false;
    }
    // c8
    {
        let getDesc = (level) =>
"c_8=7^{" + level + "}";
        let getInfo = (level) => "c_8=" +
getC8(level).toString(0);
        c8 = theory.createUpgrade(16,
currency2, new ExponentialCost(1e64,
Math.log2(1e48)));
        c8.getDescription = (_) =>
Utils.getMath(getDesc(c8.level));
        c8.getInfo = (amount) =>
Utils.getMathTo(getInfo(c8.level),
getInfo(c8.level + amount));
        c8.isAvailable = false;
    }

    {
        let getDesc = () =>
Localization.getUpgradeIncCustomDe
sc("\\text{time}", "\\text{0.1s}");
        let getInfo = () =>
Localization.getUpgradeIncCustomInf
o("\\text{time}", "\\text{0.1s}");
        clicker =
theory.createUpgrade(20, currency2,
new FreeCost());
```

```
        clicker.getDescription = (_)
=>getDesc();
        clicker.getInfo = (amount) =>
getInfo();
        clicker.bought = (amount) =>
{ prgGacha += .1 / 300 * amount *
getSkillEffect(2) };
    }
    {
        let getDesc = () => "Minigames";
        let getInfo = () => "Open the
minigame panel";
        minigame =
theory.createUpgrade(21, currency2,
new FreeCost());
        minigame.getDescription = (_)
=>getDesc();
        minigame.getInfo = (amount) =>
getInfo();
        minigame.bought = (amount) => {
            minigame.level -= amount;
            if (!isPopupOpen)
showMinigamePopup();
        };
    }

    // b3
    {
        let getDesc = (level) => "b_3=" +
getB3(level).toString(0);
        b3 = theory.createUpgrade(30,
currency3, new ExponentialCost(150,
Math.log2(2)));
        b3.getDescription = (_) =>
Utils.getMath(getDesc(b3.level));
        b3.getInfo = (amount) =>
Utils.getMathTo(getDesc(b3.level),
getDesc(b3.level + amount));
    }
    // n3
    {
        let getDesc = (level) => "n_3=" +
getN3(level).toString(0);
        n3 = theory.createUpgrade(31,
currency3, new ExponentialCost(500,
Math.log2(2)));
        n3.getDescription = (_) =>
Utils.getMath(getDesc(n3.level));
        n3.getInfo = (amount) =>
Utils.getMathTo(getDesc(n3.level),
getDesc(n3.level + amount));
    }
    // n4
    {
        let getDesc = (level) =>
"n_4=2^{" + level + "}";
        let getInfo = (level) => "n_4=" +
getN4(level).toString(0);
        n4 = theory.createUpgrade(32,
currency3, new ExponentialCost(1500,
Math.log2(1000)));
        n4.getDescription = (_) =>
Utils.getMath(getDesc(n4.level));
        n4.getInfo = (amount) =>
Utils.getMathTo(getInfo(n4.level),
getInfo(n4.level + amount));
    }
    // c9
    {
        let getDesc = (level) => "c_9={" +
(level + 1) + "}^2";
        let getInfo = (level) => "c_9=" +
getC9(level).toString(0);
        c9 = theory.createUpgrade(33,
currency3, new ExponentialCost(1e12,
Math.log2(10)));
        c9.getDescription = (_) =>
Utils.getMath(getDesc(c9.level));
        c9.getInfo = (amount) =>
Utils.getMathTo(getInfo(c9.level),
getInfo(c9.level + amount));
        c9.isAvailable = false;
    }
    // c10
    {
        let getDesc = (level) => "c_{10}
={" + (level + 1) + "}^3";
        let getInfo = (level) => "c_{10}="
+ getC10(level).toString(0);
        c10 = theory.createUpgrade(34,
currency3, new ExponentialCost(1e18,
Math.log2(25)));
        c10.getDescription = (_) =>
Utils.getMath(getDesc(c10.level));
        c10.getInfo = (amount) =>
Utils.getMathTo(getInfo(c10.level),
getInfo(c10.level + amount));
        c10.isAvailable = false;
    }
    // c11
    {
        let getDesc = (level) => "c_{11}
={" + (level + 1) + "}^4";
        let getInfo = (level) => "c_{11}=" +
getC11(level).toString(0);
        c11 = theory.createUpgrade(35,
currency3, new ExponentialCost(1e30,
Math.log2(100)));
        c11.getDescription = (_) =>
Utils.getMath(getDesc(c11.level));
        c11.getInfo = (amount) =>
Utils.getMathTo(getInfo(c11.level),
getInfo(c11.level + amount));
        c11.isAvailable = false;
    }
    // c12
    {
        let getDesc = (level) => "c_{12}
={" + (level + 1) + "}^5";
        let getInfo = (level) => "c_{12}="
+ getC12(level).toString(0);
        c12 = theory.createUpgrade(36,
currency3, new ExponentialCost(1e50,
Math.log2(1000)));
        c12.getDescription = (_) =>
Utils.getMath(getDesc(c12.level));
        c12.getInfo = (amount) =>
Utils.getMathTo(getInfo(c12.level),
getInfo(c12.level + amount));
        c12.isAvailable = false;
    }

    // d4
    {
        let getDesc = (level) => level +
"d4";
        let getInfo = (level) => "d4 \\in ["
+ level + "," + (4 * level) + "]";
        d4 = theory.createUpgrade(40,
currency3, new ExponentialCost(1e6,
1));
        d4.getDescription = (amount) =>
Utils.getMathTo(getDesc(d4.level),
getDesc(d4.level + amount));
        d4.getInfo = (amount) =>
Utils.getMathTo(getInfo(d4.level),
getInfo(d4.level + amount));
    }
    // d6
    {
        let getDesc = (level) => level +
"d6";
        let getInfo = (level) => "d6 \\in ["
+ level + "," + (6 * level) + "]";
        d6 = theory.createUpgrade(41,
currency3, new ExponentialCost(1e9,
2));
        d6.getDescription = (amount) =>
Utils.getMathTo(getDesc(d6.level),
getDesc(d6.level + amount));
        d6.getInfo = (amount) =>
Utils.getMathTo(getInfo(d6.level),
getInfo(d6.level + amount));
    }
    // d8
    {
        let getDesc = (level) => level + "d8";
        let getInfo = (level) => "d8 \\in [" + level + "," + (8 * level) + "]";
        d8 = theory.createUpgrade(42, currency3, new ExponentialCost(1e12, 3));
        d8.getDescription = (amount) => Utils.getMathTo(getDesc(d8.level), getDesc(d8.level + amount));
        d8.getInfo = (amount) => Utils.getMathTo(getInfo(d8.level), getInfo(d8.level + amount));
    }
    // d10
    {
        let getDesc = (level) => level + "d10";
        let getInfo = (level) => "d10 \\in [" + level + "," + (10 * level) + "]";
        d10 = theory.createUpgrade(43, currency3, new ExponentialCost(1e18, 4));
        d10.getDescription = (amount) => Utils.getMathTo(getDesc(d10.level), getDesc(d10.level + amount));
        d10.getInfo = (amount) => Utils.getMathTo(getInfo(d10.level), getInfo(d10.level + amount));
    }
    // d12
    {
        let getDesc = (level) => level + "d12";
        let getInfo = (level) => "d12 \\in [" + level + "," + (12 * level) + "]";
        d12 = theory.createUpgrade(44, currency3, new ExponentialCost(1e24, 5));
        d12.getDescription = (amount) => Utils.getMathTo(getDesc(d12.level), getDesc(d12.level + amount));
        d12.getInfo = (amount) => Utils.getMathTo(getInfo(d12.level), getInfo(d12.level + amount));
    }
    // d20
    {
        let getDesc = (level) => level + "d20";
        let getInfo = (level) => "d20 \\in [" + level + "," + (20 * level) + "]";
        d20 = theory.createUpgrade(45, currency3, new ExponentialCost(1e32, 6));
        d20.getDescription = (amount) => Utils.getMathTo(getDesc(d20.level), getDesc(d20.level + amount));
        d20.getInfo = (amount) => Utils.getMathTo(getInfo(d20.level), getInfo(d20.level + amount));
    }

    {
        let getDesc = () => "Boosters";
        let getInfo = () => "Open the booster panel";
        booster = theory.createUpgrade(50, currency3, new FreeCost());
        booster.getDescription = (_) => getDesc();
        booster.getInfo = (amount) => getInfo();
        booster.bought = (amount) => {
            booster.level -= amount;
            if (!isPopupOpen) showBoosterPopup();
        };
    }
    {
        let getDesc = () => "Perks";
        let getInfo = () => "Open the perk
panel";
        perk = theory.createUpgrade(51,
currency3, new FreeCost());
        perk.getDescription = (_) =>
getDesc();
        perk.getInfo = (amount) =>
getInfo();
        perk.bought = (amount) => {
            perk.level -= amount;
            if (!isPopupOpen)
showPerkPopup();
        };
    }

    ///////////////////////
    // Permanent Upgrades

    permPub =
theory.createPublicationUpgrade(0,
currency, 1e10);
    permAll =
theory.createBuyAllUpgrade(1,
currency, 1e13);
    permAuto =
theory.createAutoBuyerUpgrade(2,
currency, 1e30);

    {
        perm1 =
theory.createPermanentUpgrade(3,
currency2, new ConstantCost(100));
        perm1.maxLevel = 1;
        perm1.getDescription = (amount)
=>
Localization.getUpgradeMultCustomD
esc("\\rho_2 \\text{ gain}", "r_5 + 1");
        perm1.getInfo = (amount) =>
Localization.getUpgradeMultCustomIn
```

```
fo("\\rho_2 \\text{ gain}", "r_5 + 1");
        perm1.bought = (_) =>
{ theory.invalidateSecondaryEquation(
); };
    }
    {
        perm2 =
theory.createPermanentUpgrade(4,
currency2, new
ConstantCost(10000));
        perm2.maxLevel = 1;
        perm2.getDescription = (amount)
=>
Localization.getUpgradeMultCustomD
esc("\\rho_1 \\text{ gain}", "r_6 + 1");
        perm2.getInfo = (amount) =>
Localization.getUpgradeMultCustomIn
fo("\\rho_1 \\text{ gain}", "r_6 + 1");
    }
    {
        perm3 =
theory.createPermanentUpgrade(5,
currency2, new
ConstantCost(1000000));
        perm3.maxLevel = 1;
        perm3.getDescription = (amount)
=>
Localization.getUpgradeMultCustomD
esc("c_1", "r_7 + 1");
        perm3.getInfo = (amount) =>
Localization.getUpgradeMultCustomIn
fo("c_1", "r_7 + 1");
    }
    {
        perm4 =
theory.createPermanentUpgrade(6,
currency2, new
ConstantCost(1000000000));
        perm4.maxLevel = 1;
        perm4.getDescription = (amount)
=>
```

```
Localization.getUpgradeMultCustomDesc("b_1", "r_8 + 1");
        perm4.getInfo = (amount) =>
Localization.getUpgradeMultCustomInfo("b_1", "r_8 + 1");
    }

    {
        perm5 =
theory.createPermanentUpgrade(7,
currency3, new ExponentialCost(100,
2));
        perm5.getDescription = (amount)
=>
Localization.getUpgradeMultCustomDesc("\\rho_3 \\text{ gain}", "100^{" +
perm5.level + "}");
        perm5.getInfo = (amount) =>
Localization.getUpgradeMultCustomInfo("\\rho_1 \\text{ gain}",
getPerm5(perm5.level));
        perm5.bought = (_) =>
{ theory.invalidateSecondaryEquation(
); };
    }
    {
        perm6 =
theory.createPermanentUpgrade(8,
currency3, new ExponentialCost(1000,
5));
        perm6.getDescription = (amount)
=>
Localization.getUpgradeMultCustomDesc("\\rho_2 \\text{ gain}", "5^{" +
perm6.level + "}");
        perm6.getInfo = (amount) =>
Localization.getUpgradeMultCustomInfo("\\rho_2 \\text{ gain}",
getPerm6(perm6.level));
        perm6.bought = (_) =>
{ theory.invalidateSecondaryEquation(
); };
    }
    {
        perm7 =
theory.createPermanentUpgrade(9,
currency3, new
ExponentialCost(10000, 10));
        perm7.getDescription = (amount)
=>
Localization.getUpgradeMultCustomD
esc("\\text{pub. multi.}", "20^{" +
perm7.level + "}");
        perm7.getInfo = (amount) =>
Localization.getUpgradeMultCustomIn
fo("\\text{publication multiplier}",
getPerm7(perm7.level));
        perm7.bought = (_) =>
{ theory.invalidateSecondaryEquation(
); };
    }
    {
        perm8 =
theory.createPermanentUpgrade(10,
currency3, new
ExponentialCost(100000, 20));
        perm8.getDescription = (amount)
=>
Localization.getUpgradeMultCustomD
esc("\\rho_3 \\text{ gain}", "2^{" +
perm8.level + "}");
        perm8.getInfo = (amount) =>
Localization.getUpgradeMultCustomIn
fo("\\rho_3 \\text{ gain}",
getPerm8(perm8.level));
        perm8.bought = (_) =>
{ theory.invalidateSecondaryEquation(
); };
    }

    {
        perm9 =
theory.createPermanentUpgrade(17,
currency3, new ConstantCost(1e12));
        perm9.maxLevel = 1;
        perm9.getDescription = (amount)
=>
Localization.getUpgradeMultCustomD
esc("r_5", "q_5 + 1");
        perm9.getInfo = (amount) =>
Localization.getUpgradeMultCustomIn
fo("r_5", "q_5 + 1");
    }
    {
        perm10 =
theory.createPermanentUpgrade(18,
currency3, new ConstantCost(1e20));
        perm10.maxLevel = 1;
        perm10.getDescription =
(amount) =>
Localization.getUpgradeMultCustomD
esc("r_6", "q_6 + 1");
        perm10.getInfo = (amount) =>
Localization.getUpgradeMultCustomIn
fo("r_6", "q_6 + 1");
    }
    {
        perm11 =
theory.createPermanentUpgrade(19,
currency3, new ConstantCost(1e30));
        perm11.maxLevel = 1;
        perm11.getDescription = (amount)
=>
Localization.getUpgradeMultCustomD
esc("r_7", "q_7 + 1");
        perm11.getInfo = (amount) =>
Localization.getUpgradeMultCustomIn
fo("r_7", "q_7 + 1");
    }
    {
        perm12 =
theory.createPermanentUpgrade(20,
currency3, new ConstantCost(1e45));
```

```
        perm12.maxLevel = 1;
        perm12.getDescription =
(amount) =>
Localization.getUpgradeMultCustomD
esc("r_8", "q_8 + 1");
        perm12.getInfo = (amount) =>
Localization.getUpgradeMultCustomIn
fo("r_8", "q_8 + 1");
    }

    {
        d4Master =
theory.createPermanentUpgrade(11,
currency3, new FreeCost());
        d4Master.getDescription =
(amount) => "$d4$ Mastery";
        d4Master.getInfo = (amount) =>
"\\textsf{" + ["MT", "EX", "DL", "SD",
"CL", "CT"].filter((x, i) =>
diceMasteries[i] == "#").join(" ") + "}";
        d4Master.maxLevel = 6;
        d4Master.bought = (amount) => {
            if (tSkill == 0) {
                tSkill = null;
            } else {
                d4Master.level -= amount;
                showDiceMasteryPopup(0);
            }
        }
    }
    {
        d6Master =
theory.createPermanentUpgrade(12,
currency3, new FreeCost());
        d6Master.getDescription =
(amount) => "$d6$ Mastery";
        d6Master.getInfo = (amount) =>
"\\textsf{" + ["MT", "EX", "DL", "SD",
"CL", "CT"].filter((x, i) =>
diceMasteries[i + 6] == "#").join(" ") +
"}";
```

```
        d6Master.maxLevel = 6;
        d6Master.bought = (amount) => {
            if (tSkill == 1) {
                tSkill = null;
            } else {
                d6Master.level -= amount;
                showDiceMasteryPopup(1);
            }
        }
    }
    {
        d8Master =
theory.createPermanentUpgrade(13,
currency3, new FreeCost());
        d8Master.getDescription =
(amount) => "$d8$ Mastery";
        d8Master.getInfo = (amount) =>
"\\textsf{" + ["MT", "EX", "DL", "SD",
"CL", "CT"].filter((x, i) =>
diceMasteries[i + 12] == "#").join(" ")
+ "}";
        d8Master.maxLevel = 6;
        d8Master.bought = (amount) => {
            if (tSkill == 2) {
                tSkill = null;
            } else {
                d8Master.level -= amount;
                showDiceMasteryPopup(2);
            }
        }
    }
    {
        d10Master =
theory.createPermanentUpgrade(14,
currency3, new FreeCost());
        d10Master.getDescription =
(amount) => "$d10$ Mastery";
        d10Master.getInfo = (amount) =>
"\\textsf{" + ["MT", "EX", "DL", "SD",
"CL", "CT"].filter((x, i) =>
diceMasteries[i + 18] == "#").join(" ")
+ "}";
        d10Master.maxLevel = 6;
        d10Master.bought = (amount) =>
{
            if (tSkill == 3) {
                tSkill = null;
            } else {
                d10Master.level -= amount;
                showDiceMasteryPopup(3);
            }
        }
    }
    {
        d12Master =
theory.createPermanentUpgrade(15,
currency3, new FreeCost());
        d12Master.getDescription =
(amount) => "$d12$ Mastery";
        d12Master.getInfo = (amount) =>
"\\textsf{" + ["MT", "EX", "DL", "SD",
"CL", "CT"].filter((x, i) =>
diceMasteries[i + 24] == "#").join(" ")
+ "}";
        d12Master.maxLevel = 6;
        d12Master.bought = (amount) =>
{
            if (tSkill == 4) {
                tSkill = null;
            } else {
                d12Master.level -= amount;
                showDiceMasteryPopup(4);
            }
        }
    }
    {
        d20Master =
theory.createPermanentUpgrade(16,
currency3, new FreeCost());
        d20Master.getDescription =
(amount) => "$d20$ Mastery";
        d20Master.getInfo = (amount) =>
```

```
"\\textsf{" + ["MT", "EX", "DL", "SD",
"CL", "CT"].filter((x, i) =>
diceMasteries[i + 30] == "#").join(" ")
+ "}";
        d20Master.maxLevel = 6;
        d20Master.bought = (amount) =>
{
            if (tSkill == 5) {
                tSkill = null;
            } else {
                d20Master.level -= amount;
                showDiceMasteryPopup(5);
            }
        }
    }

    {
        perm13 =
theory.createPermanentUpgrade(21,
currency, new
ExponentialCost(BigNumber.from("e6
750"), 200));
        perm13.maxLevel = 100;
        perm13.getDescription =
(amount) =>
Localization.getUpgradeMultCustomD
esc("\\rho_1", (perm13.level + 1) +
"^{Score}");
        perm13.getInfo = (amount) =>
Localization.getUpgradeMultCustomIn
fo("\\rho_1", getPerm13(perm13.level)
+ "\\rightarrow" +
getPerm13(perm13.level + amount));
    }
    {
        perm14 =
theory.createPermanentUpgrade(22,
currency2, new
ExponentialCost(1e275, 100));
        perm14.maxLevel = 50;
        perm14.getDescription =
(amount) =>
Localization.getUpgradeMultCustomD
esc("\\rho_2", (perm14.level + 1) +
"^{Score/2}");
        perm14.getInfo = (amount) =>
Localization.getUpgradeMultCustomIn
fo("\\rho_2", getPerm14(perm14.level)
+ "\\rightarrow" +
getPerm14(perm14.level + amount));
    }
    {
        perm15 =
theory.createPermanentUpgrade(23,
currency3,  new
ExponentialCost(1e25, 50));
        perm15.maxLevel = 25;
        perm15.getDescription =
(amount) =>
Localization.getUpgradeMultCustomD
esc("\\rho_3", (perm15.level + 1) +
"^{Score/4}");
        perm15.getInfo = (amount) =>
Localization.getUpgradeMultCustomIn
fo("\\rho_3", getPerm15(perm15.level)
+ "\\rightarrow" +
getPerm15(perm15.level + amount));
    }

    for (let sd in skillData) {
        let data = skillData[sd];
        skills[sd] =
theory.createPermanentUpgrade(1000
+ sd, currency2, new FreeCost);
        skills[sd].getDescription =
data.name;
        if (data.maxLevel)
skills[sd].maxLevel = data.maxLevel;
        skills[sd].getInfo = () => {
            let eff = getSkillEffect(sd)
            return
data.info().replace("{eff}", eff
instanceof BigNumber ?
eff.toString(0) :
                eff.toLocaleString("en-US",
{ minimumFractionDigits: 0,
maximumFractionDigits: 2 }));
        }
        skills[sd].bought = (amount) => {
            if (tSkill == sd) {
                tSkill = null;
            } else {
                skills[sd].level -= amount;
                if (!isPopupOpen)
showSkillPopup(sd);
            }
        }
        skillExp[sd] = 0;
    }

    ////////////////////
    // Milestone Upgrades

    theory.setMilestoneCost(new
CompositeCost(9, new LinearCost(25,
30), new CompositeCost(94, new
LinearCost(295, 25), new
ExponentialCost(2750,
.13750352375))));

    {
        prgSpeed =
theory.createMilestoneUpgrade(0, 3);
        prgSpeed.getDescription = (_) =>
Localization.getUpgradeIncCustomDe
sc("\\text{speed}", "100\\%");
        prgSpeed.getInfo = (_) =>
Localization.getUpgradeIncCustomInf
o("\\text{speed}", "100\\%");
        prgSpeed.canBeRefunded = (_)
=> (cTerms.level == 0 ||
prgSpeed.level > 1) && (qTerms.level
== 0 || prgSpeed.level > 2) &&
(cur2Unlock.level == 0);
        prgSpeed.boughtOrRefunded =
(_) => { updateAvailability(); };
    }
    {
        rTerms =
theory.createMilestoneUpgrade(1, 2);
        rTerms.getDescription = (_) =>
Localization.getUpgradeUnlockDesc(r
Terms.level == 0 ? "r_3" : "r_4");
        rTerms.getInfo = (_) =>
Localization.getUpgradeUnlockInfo(rT
erms.level == 0 ? "r_3" : "r_4");
        rTerms.canBeRefunded = (_) =>
(cur2Unlock.level == 0);
        rTerms.boughtOrRefunded = (_)
=>
{ theory.invalidatePrimaryEquation();
theory.invalidateQuaternaryValues();
updateAvailability(); };
    }
    {
        cTerms =
theory.createMilestoneUpgrade(2, 2);
        cTerms.getDescription = (_) =>
Localization.getUpgradeUnlockDesc(c
Terms.level == 0 ? "c_3" : "c_4");
        cTerms.getInfo = (_) =>
Localization.getUpgradeUnlockInfo(cT
erms.level == 0 ? "c_3" : "c_4");
        cTerms.canBeRefunded = (_) =>
(cur2Unlock.level == 0);
        cTerms.boughtOrRefunded = (_)
=>
{ theory.invalidatePrimaryEquation();
updateAvailability(); };
        cTerms.isAvailable = false;
    }
    {
        qTerms =
theory.createMilestoneUpgrade(3, 4);
```

```
        qTerms.getDescription = (_) =>
Localization.getUpgradeUnlockDesc("
q_" + Math.min(qTerms.level + 1, 4));
        qTerms.getInfo = (_) =>
Localization.getUpgradeUnlockInfo("q
_" + Math.min(qTerms.level + 1, 4));
        qTerms.canBeRefunded = (_) =>
(cur2Unlock.level == 0);
        qTerms.boughtOrRefunded = (_)
=>
{ theory.invalidatePrimaryEquation();
theory.invalidateSecondaryEquation();
updateAvailability(); };
        qTerms.isAvailable = false;
    }
    {
        cur2Unlock =
theory.createMilestoneUpgrade(4, 1);
        cur2Unlock.getDescription = (_)
=>
Localization.getUpgradeUnlockDesc("\
\rho_2");
        cur2Unlock.getInfo = (_) =>
Localization.getUpgradeUnlockInfo("\
\rho_2");
        cur2Unlock.canBeRefunded = (_)
=> false;
        cur2Unlock.isAvailable = false;
    }
    {
        gachaUnlock =
theory.createMilestoneUpgrade(5, 1);
        gachaUnlock.getDescription = (_)
=>
Localization.getUpgradeUnlockDesc("\
\ominus");
        gachaUnlock.getInfo = (_) =>
Localization.getUpgradeUnlockInfo("\
\ominus");
        gachaUnlock.canBeRefunded =
(_) => false;
```

```
        gachaUnlock.isAvailable = false;
    }
    {
        gachaValue =
theory.createMilestoneUpgrade(6, 15);
        gachaValue.getDescription = (_)
=> "$\\uparrow \\bigstar \\text{ pull
rates}$";
        gachaValue.getInfo = (_) => "$P(\
\bigstar_i) = " + (5 - gachaValue.level
* .2).toFixed(1) + "P(\\bigstar_{i+1})$";
        gachaValue.canBeRefunded = (_)
=> false;
        gachaValue.isAvailable = false;
    }
    {
        gachaBulk =
theory.createMilestoneUpgrade(7, 74);
        gachaBulk.getDescription = (_)
=> "$\\uparrow \\text{max bulk }\
\ominus\\text{ pull}$";
        gachaBulk.getInfo = (_) => "$\
\text{Increases max bulk }\\ominus\
\text{ pull}$";
        gachaBulk.boughtOrRefunded =
(amount) =>
{ theory.invalidateSecondaryEquation(
) };
        gachaBulk.canBeRefunded = (_)
=> false;
        gachaBulk.isAvailable = false;
    }
    {
        cur3Unlock =
theory.createMilestoneUpgrade(8, 1);
        cur3Unlock.getDescription = (_)
=>
Localization.getUpgradeUnlockDesc("\
\rho_3");
        cur3Unlock.getInfo = (_) =>
Localization.getUpgradeUnlockInfo("\
\rho_3");
        cur3Unlock.canBeRefunded = (_)
=> false;
        cur3Unlock.isAvailable = false;
    }
    {
        prg3Speed =
theory.createMilestoneUpgrade(9, 6);
        prg3Speed.getDescription = (_)
=>
Localization.getUpgradeIncCustomDe
sc("\\rho_3\\text{ speed}", "50\\%");
        prg3Speed.getInfo = (_) =>
Localization.getUpgradeIncCustomInf
o("\\rho_3\\text{ speed}", "50\\%");
        prg3Speed.canBeRefunded = (_)
=> (diceUnlock.level == 0 ||
prg3Speed.level > 3) &&
(spUnlock.level == 0 ||
prg3Speed.level > 6);
        prg3Speed.boughtOrRefunded =
(_) => { updateAvailability(); };
        prg3Speed.isAvailable = false;
    }
    {
        qTerms3 =
theory.createMilestoneUpgrade(10, 2);
        qTerms3.getDescription = (_) =>
Localization.getUpgradeUnlockDesc("
q_" + Math.min(qTerms3.level + 7, 8)
+ "\\text{ and a new upgrade}");
        qTerms3.getInfo = (_) =>
Localization.getUpgradeUnlockInfo("q
_" + Math.min(qTerms3.level + 7, 8) +
"\\text{ and a new upgrade}");
        qTerms3.canBeRefunded = (_) =>
(diceUnlock.level == 0 || qTerms3.level
> 2);
        qTerms3.boughtOrRefunded = (_)
=>
{ theory.invalidatePrimaryEquation();
```

```
theory.invalidateQuaternaryValues();
updateAvailability(); };
        qTerms3.isAvailable = false;
    }
    {
        diceUnlock =
theory.createMilestoneUpgrade(11, 1);
        diceUnlock.getDescription = (_)
=>
Localization.getUpgradeUnlockDesc("\
\boxdot");
        diceUnlock.getInfo = (_) =>
Localization.getUpgradeUnlockInfo("\
\boxdot");
        diceUnlock.boughtOrRefunded =
(_) => { updateAvailability(); };
        diceUnlock.canBeRefunded = (_)
=> false;
        diceUnlock.isAvailable = false;
    }
    {
        spUnlock =
theory.createMilestoneUpgrade(12, 1);
        spUnlock.getDescription = (_) =>
Localization.getUpgradeUnlockDesc("
$\\textsf{SP}$");
        spUnlock.getInfo = (_) =>
Localization.getUpgradeUnlockInfo("$
the ability to draw \\textsf{SP} from $\
\ominus$ $");
        spUnlock.boughtOrRefunded =
(_) => { updateAvailability(); };
        spUnlock.canBeRefunded = (_)
=> false;
        spUnlock.isAvailable = false;
    }
    {
        gachaValue2 =
theory.createMilestoneUpgrade(13, 4);
        gachaValue2.getDescription = (_)
=>
```

```
Localization.getUpgradeIncCustomDesc("\\star\\text{ from }\\ominus", "1");
        gachaValue2.getInfo = (_) =>
Localization.getUpgradeIncCustomInfo("\\star\\text{ from }\\ominus", "1");
        gachaValue2.boughtOrRefunded
= (_) => { updateAvailability(); };
        gachaValue2.canBeRefunded =
(_) => rpgUnlock.level == 0;
        gachaValue2.isAvailable = false;
    }
    {
        rpgUnlock =
theory.createMilestoneUpgrade(14, 1);
        rpgUnlock.getDescription = (_)
=>
Localization.getUpgradeUnlockDesc("
$the \\textsf{RPG}$");
        rpgUnlock.getInfo = (_) =>
Localization.getUpgradeUnlockInfo("$
the \"\\textsf{R}isky \\textsf{P}robation
(on stocasticity researchin\\textsf{G})
\"$");
        rpgUnlock.boughtOrRefunded =
(_) => { updateAvailability(); };
        rpgUnlock.canBeRefunded = (_)
=> false;
        rpgUnlock.isAvailable = false;
    }
    {
        atkBoost =
theory.createMilestoneUpgrade(15,
10);
        atkBoost.getDescription = (_) =>
Localization.getUpgradeIncCustomDe
sc("$\\textsf{ATK} power$", "15\\% \
\text{ (additive)}");
        atkBoost.getInfo = (_) =>
Localization.getUpgradeIncCustomInf
o("$\\textsf{ATK} power$", "15\\% \
\text{ (additive)}");
```

```
        atkBoost.boughtOrRefunded = (_)
=> { updateAvailability(); };
        atkBoost.isAvailable = false;
    }
    {
        workNerf =
theory.createMilestoneUpgrade(16,
10);
        workNerf.getDescription = (_) =>
Localization.getUpgradeDecCustomDe
sc("$work$", "5\\% \
\text{ (multiplicative)}");
        workNerf.getInfo = (_) =>
Localization.getUpgradeDecCustomInf
o("$work$", "5\\% \
\text{ (multiplicative)}");
        workNerf.boughtOrRefunded =
(_) => { updateAvailability(); };
        workNerf.isAvailable = false;
    }
    {
        diceBoost =
theory.createMilestoneUpgrade(17,
10);
        diceBoost.getDescription = (_) =>
Localization.getUpgradeIncCustomDe
sc("\\boxdot \\text{ gain}", "10\\% \
\text{ (multiplicative)}");
        diceBoost.getInfo = (_) =>
Localization.getUpgradeIncCustomInf
o("\\boxdot \\text{ gain}", "10\\% \
\text{ (multiplicative)}");
        diceBoost.boughtOrRefunded =
(_) => { updateAvailability(); };
        diceBoost.isAvailable = false;
    }
    {
        diceBoost2 =
theory.createMilestoneUpgrade(18,
10);
        diceBoost2.getDescription = (_)
```

```
=> Localization.getUpgradeIncCustomDesc("\\dot{\\boxdot}$ gain$", "10\\% \\text{ (multiplicative)}");
        diceBoost2.getInfo = (_) => Localization.getUpgradeIncCustomInfo("\\dot{\\boxdot}$ gain$", "10\\% \\text{ (multiplicative)}");
        diceBoost2.boughtOrRefunded = (_) => { updateAvailability(); };
        diceBoost2.isAvailable = false;
    }


    ///////////////
    // Achievements

    ac1 = theory.createAchievementCategory(0, "Progression");
    theory.createAchievement(0, ac1, "And So It Begins", "Buy the first upgrade.", () => b1.level > 0);
    theory.createAchievement(1, ac1, "And So It Begins... Again", "Publish your theory.", () => theory.publicationMultiplier > 1);
    theory.createAchievement(2, ac1, "Under the Hood", "Unlock a new currency.", () => cur2Unlock.level > 0);
    theory.createAchievement(3, ac1, "Is This Gambling?", "Unlock ⊝.", () => gachaUnlock.level > 0);
    theory.createAchievement(4, ac1, "Gotta Collect 'em All", "Unlock all 13 normal and EX ⊝ skills.", () => {
        for (let a = 0; a < 13; a++) if (skills[a].level == 0) return false;
        return true;
    });
    theory.createAchievement(5, ac1,
"Down the Rabbit Hole", "Unlock
another new currency.", () =>
cur3Unlock.level > 0);
    theory.createAchievement(6, ac1,
"Role-Playing Scientist", "Unlock ⊡.",
() => diceUnlock.level > 0);
    theory.createAchievement(7, ac1,
"Gotta Collect More of 'em", "Unlock
all 8 SP ⊝ skills.", () => {
        for (let a = 13; a < 21; a++) if
(skills[a].level == 0) return false;
        return true;
    });
    theory.createAchievement(8, ac1,
"Progress Bars", "Unlock the RPG.", ()
=> rpgUnlock.level > 0);
    theory.createAchievement(9, ac1,
"Master of a Die", "Fully master one
die.", () => {
        for (let a = 0; a < 6; a++) if
(diceMasteries.substring(a * 6, a * 6 +
6) == "######") return true;
        return false;
    });
    theory.createAchievement(10, ac1,
"Master of All Dice", "Fully master all
the dice.", () => {
        return diceMasteries ==
"############################
########"
    });

    ac2 =
theory.createAchievementCategory(1,
"Net Worth");
    theory.createAchievement(50, ac2,
"One Million is a lot", "Reach one
million τ.", () => theory.tau >= 1e6,
        () => ((theory.tau +
1).log10().toNumber() / 6));
    theory.createAchievement(51, ac2,
"Costs Are Not Divided", "Reach one trillion τ.", () => theory.tau >= 1e12,
        () => ((theory.tau + 1).log10().toNumber() / 12));
    theory.createAchievement(52, ac2, "A Small Discovery", "Reach 1e25 τ.", () => theory.tau >= 1e25,
        () => ((theory.tau + 1).log10().toNumber() / 25));
    theory.createAchievement(53, ac2, "Numbers Growing So Fast", "Reach 1e50 τ.", () => theory.tau >= 1e50,
        () => ((theory.tau + 1).log10().toNumber() / 50));
    theory.createAchievement(54, ac2, "Art of Googology", "Reach 1e100 τ.", () => theory.tau >= 1e100,
        () => ((theory.tau + 1).log10().toNumber() / 100));
    theory.createAchievement(55, ac2, "Technical Feat", "Reach 1.798e308 τ.", () => theory.tau >= Number.MAX_VALUE,
        () => ((theory.tau + 1).log10().toNumber() / Math.log10(Number.MAX_VALUE)));
    theory.createAchievement(56, ac2, "Accelerating Development", "Reach 1e1000 τ.", () => theory.tau >= BigNumber.from("e1000"),
        () => ((theory.tau + 1).log10().toNumber() / 1000));
    theory.createAchievement(57, ac2, "Millillion is Not a Typo", "Reach 1e3003 τ.", () => theory.tau >= BigNumber.from("e3003"),
        () => ((theory.tau + 1).log10().toNumber() / 3003));
    theory.createAchievement(59, ac2, "Insane Mindset", "Reach 1e7500 τ.", () => theory.tau >=
BigNumber.from("e7500"),
        () => ((theory.tau +
1).log10().toNumber() / 7500));
    theory.createAchievement(60, ac2,
"Work from the Gods", "Reach
1e15000 τ.", () => theory.tau >=
BigNumber.from("e15000"),
        () => ((theory.tau +
1).log10().toNumber() / 15000));
    theory.createAchievement(61, ac2,
"Humanity's Knowledge", "Reach
1e30000 τ.", () => theory.tau >=
BigNumber.from("e30000"),
        () => ((theory.tau +
1).log10().toNumber() / 30000));
    theory.createAchievement(62, ac2,
"Tomorrowland", "Reach 1e60000 τ.",
() => theory.tau >=
BigNumber.from("e60000"),
        () => ((theory.tau +
1).log10().toNumber() / 60000));
    theory.createAchievement(63, ac2,
"Library of the Galaxy", "Reach
1e100000 τ.", () => theory.tau >=
BigNumber.from("e100000"),
        () => ((theory.tau +
1).log10().toNumber() / 100000));

    ac3 =
theory.createAchievementCategory(2,
"Publicity");
    theory.createAchievement(100, ac3,
"Nobody", "Reach 100 publication
multiplier.", () =>
theory.publicationMultiplier >= 100,
        () =>
((theory.publicationMultiplier +
1).log10().toNumber() /
Math.log10(101)));
    theory.createAchievement(101, ac3,
"A Small Name", "Reach 1 million
```

```
publication multiplier.", () =>
theory.publicationMultiplier >= 1e6,
        () =>
((theory.publicationMultiplier +
1).log10().toNumber() / 6));
    theory.createAchievement(102, ac3,
"Gaining Attraction", "Reach 1 trillion
publication multiplier.", () =>
theory.publicationMultiplier >= 1e12,
        () =>
((theory.publicationMultiplier +
1).log10().toNumber() / 12));
    theory.createAchievement(103, ac3,
"City's Name", "Reach 1e25
publication multiplier.", () =>
theory.publicationMultiplier >= 1e25,
        () =>
((theory.publicationMultiplier +
1).log10().toNumber() / 25));
    theory.createAchievement(104, ac3,
"Country's Name", "Reach 1e50
publication multiplier.", () =>
theory.publicationMultiplier >= 1e50,
        () =>
((theory.publicationMultiplier +
1).log10().toNumber() / 50));
    theory.createAchievement(105, ac3,
"Instantly Recognizable", "Reach 1e75
publication multiplier.", () =>
theory.publicationMultiplier >= 1e75,
        () =>
((theory.publicationMultiplier +
1).log10().toNumber() / 75));
    theory.createAchievement(106, ac3,
"Gambilician of the Year", "Reach
1e100 publication multiplier.", () =>
theory.publicationMultiplier >= 1e100,
        () =>
((theory.publicationMultiplier +
1).log10().toNumber() / 100));
    theory.createAchievement(107, ac3,
"Everyone Now Knows You", "Reach 1e150 publication multiplier.", () => theory.publicationMultiplier >= 1e150,
        () => ((theory.publicationMultiplier + 1).log10().toNumber() / 150));
    theory.createAchievement(108, ac3, "Gambilician of the Decade", "Reach 1e200 publication multiplier.", () => theory.publicationMultiplier >= 1e200,
        () => ((theory.publicationMultiplier + 1).log10().toNumber() / 200));
    theory.createAchievement(109, ac3, "Popularity Off the Scale", "Reach 1.798e308 publication multiplier.", () => theory.publicationMultiplier >= Number.MAX_VALUE,
        () => ((theory.publicationMultiplier + 1).log10().toNumber() / Math.log10(Number.MAX_VALUE)));
    theory.createAchievement(110, ac3, "Gambilician of the Century", "Reach 1e500 publication multiplier.", () => theory.publicationMultiplier >= BigNumber.from("e500"),
        () => ((theory.publicationMultiplier + 1).log10().toNumber() / 500));
    theory.createAchievement(111, ac3, "You're in the History Books", "Reach 1e750 publication multiplier.", () => theory.publicationMultiplier >= BigNumber.from("e750"),
        () => ((theory.publicationMultiplier + 1).log10().toNumber() / 750));
    theory.createAchievement(112, ac3, "Gambilician of the Millenium", "Reach 1e1000 publication multiplier.", () =>
theory.publicationMultiplier >=
BigNumber.from("e1000"),
        () =>
((theory.publicationMultiplier +
1).log10().toNumber() / 1000));
    theory.createAchievement(113, ac3,
"The Aliens Know You", "Reach
1e2000 publication multiplier.", () =>
theory.publicationMultiplier >=
BigNumber.from("e2000"),
        () =>
((theory.publicationMultiplier +
1).log10().toNumber() / 2000));
    theory.createAchievement(114, ac3,
"Gambilician of the Eon", "Reach
1e4000 publication multiplier.", () =>
theory.publicationMultiplier >=
BigNumber.from("e4000"),
        () =>
((theory.publicationMultiplier +
1).log10().toNumber() / 4000));

    ac4 =
theory.createAchievementCategory(3,
"Gachapons");
    theory.createAchievement(150, ac4,
"Some Collectibles", "Get 5 total Θ.",
() => gachaTotal >= 5,
        () => (gachaTotal.toNumber() /
5));
    theory.createAchievement(151, ac4,
"Pocket Toys", "Get 10 total Θ.", () =>
gachaTotal >= 10,
        () => (gachaTotal.toNumber() /
10));
    theory.createAchievement(152, ac4,
"Colecting is Fun", "Get 25 total Θ.", ()
=> gachaTotal >= 25,
        () => (gachaTotal.toNumber() /
25));
    theory.createAchievement(153, ac4,
"Stars, Come Out!", "Get 50 total ⊝.",
() => gachaTotal >= 50,
        () => (gachaTotal.toNumber() /
50));
    theory.createAchievement(154, ac4,
"Fortune on Figures", "Get 100 total
⊝.", () => gachaTotal > 100,
        () => (gachaTotal.toNumber() /
100));
    theory.createAchievement(155, ac4,
"Am I Addicted?", "Get 250 total ⊝.", ()
=> gachaTotal >= 250,
        () => (gachaTotal.toNumber() /
250));
    theory.createAchievement(156, ac4,
"Fallen Stars", "Get 500 total ⊝.", () =>
gachaTotal >= 500,
        () => (gachaTotal.toNumber() /
500));
    theory.createAchievement(157, ac4,
"Fallen Galaxies", "Get 1000 total ⊝.",
() => gachaTotal >= 1000,
        () => (gachaTotal.toNumber() /
1000));
    theory.createAchievement(158, ac4,
"Fallen Universe", "Get 2500 total ⊝.",
() => gachaTotal >= 2500,
        () => (gachaTotal.toNumber() /
2500));
    theory.createAchievement(159, ac4,
"exp5000", "Get 5000 total ⊝.", () =>
gachaTotal >= 5000,
        () => (gachaTotal.toNumber() /
5000));
    theory.createAchievement(160, ac4,
"Really Big Toy Collection", "Get
10000 total ⊝.", () => gachaTotal >=
10000,
        () => (gachaTotal.toNumber() /
10000));
    theory.createAchievement(161, ac4,
"All of 'em", "Get 25000 total ⊝.", ()
=> gachaTotal >= 25000,
        () => (gachaTotal.toNumber() /
25000));
    theory.createAchievement(162, ac4,
"And It's All Free", "Get 50000 total
⊝.", () => gachaTotal >= 50000,
        () => (gachaTotal.toNumber() /
50000));
    theory.createAchievement(163, ac4,
"Dedicated Grinder", "Get 100000
total ⊝.", () => gachaTotal >= 100000,
        () => (gachaTotal.toNumber() /
100000));
    theory.createAchievement(164, ac4,
"OK That's Enough", "Get 250000
total ⊝.", () => gachaTotal >= 250000,
        () => (gachaTotal.toNumber() /
250000));

    ac5 =
theory.createAchievementCategory(4,
"Minigames");
    theory.createAchievement(200, ac5,
"Just a Small Break", "Reach a total
best score of 1000 on all minigames.",
() => hiscores[0] + hiscores[1] +
hiscores[2] >= 1000,
        () => (hiscores[0] + hiscores[1] +
hiscores[2]) / 1000);
    theory.createAchievement(201, ac5,
"Casual Mathematician", "Reach a
total best score of 2500 on all
minigames.", () => hiscores[0] +
hiscores[1] + hiscores[2] >= 2500,
        () => (hiscores[0] + hiscores[1] +
hiscores[2]) / 2500);
    theory.createAchievement(202, ac5,
"Game Theorist", "Reach a total best
score of 5000 on all minigames.", () =>
hiscores[0] + hiscores[1] + hiscores[2]
```

```
>= 5000,
        () => (hiscores[0] + hiscores[1] +
hiscores[2]) / 5000);
    theory.createAchievement(203, ac5,
"IQ > 9000", "Reach a total best score
of 9001 on all minigames.", () =>
hiscores[0] + hiscores[1] + hiscores[2]
>= 9001,
        () => (hiscores[0] + hiscores[1] +
hiscores[2]) / 9001);
    theory.createAchievement(204, ac5,
"Polyominoes", "Reach a best score of
1000 on the Block Puzzle minigame.",
() => hiscores[0] >= 1000, () =>
hiscores[0] / 1000);
    theory.createAchievement(205, ac5,
"Block Breaker Master", "Reach a best
score of 2500 on the Block Puzzle
minigame.", () => hiscores[0] >= 2500,
() => hiscores[0] / 2500);
    theory.createAchievement(206, ac5,
"They Aren't Dice", "Reach a best
score of 1000 on the Domino Puzzle
minigame.", () => hiscores[1] >= 1000,
() => hiscores[1] / 1000);
    theory.createAchievement(207, ac5,
"Unknockable", "Reach a best score of
2500 on the Domino Puzzle
minigame.", () => hiscores[1] >= 2500,
() => hiscores[1] / 2500)

    ac6 =
theory.createAchievementCategory(5,
"Dice");
    theory.createAchievement(250, ac6,
"Totally Not Gambling", "Reach 1000
⚀.", () => dicePoints >= 1000,
        () => (dicePoints.toNumber() /
1000));
    theory.createAchievement(251, ac6,
"1km²", "Reach 1000000 ⚀.", () =>
```

```
dicePoints >= 1000000,
        () => (dicePoints.toNumber() /
1000000));
    theory.createAchievement(252, ac6,
"High Roller", "Reach 1000000000 ⊡.",
() => dicePoints >= 1e9,
        () => (dicePoints.toNumber() /
1e9));
    theory.createAchievement(253, ac6,
"How Many Digits Is That?", "Reach
1000000000000 ⊡.", () => dicePoints
>= 1e12,
        () => (dicePoints.toNumber() /
1e12));
    theory.createAchievement(254, ac6,
"Makai Senki", "Reach
1000000000000000 ⊡.", () =>
dicePoints >= 1e15,
        () => (dicePoints.toNumber() /
1e15));
    theory.createAchievement(255, ac6,
"Auto-Roller", "Reach 1 ⊡/s.", () =>
dicePoints2 >= 1,
        () => (dicePoints2.toNumber() /
1));
    theory.createAchievement(256, ac6,
"Snake Eyes", "Reach 1000 ⊡/s.", () =>
dicePoints2 >= 1000,
        () => (dicePoints2.toNumber() /
1000));
    theory.createAchievement(257, ac6,
"Dice Factory", "Reach 1000000 ⊡/s.",
() => dicePoints2 >= 1000000,
        () => (dicePoints2.toNumber() /
1000000));
    theory.createAchievement(258, ac6,
"Lightspeed", "Reach 1000000000 ⊡/
s.", () => dicePoints2 >= 1e9,
        () => (dicePoints2.toNumber() /
1e9));
    theory.createAchievement(259, ac6,
"Warping Rollin'", "Reach
1000000000000 ⊡/s.", () =>
dicePoints2 >= 1e12,
        () => (dicePoints2.toNumber() /
1e12));

    ac7 =
theory.createAchievementCategory(6,
"RPG");
    theory.createAchievement(300, ac7,
"Level 1 Crook", "Reach an RPG score
of 10.", () => rpgScore >= 10,
        () => (rpgScore / 10));
    theory.createAchievement(301, ac7,
"Half Way There", "Reach an RPG
score of 25.", () => rpgScore >= 25,
        () => (rpgScore / 25));
    theory.createAchievement(302, ac7,
"You Beat The Theory!", "Reach an
RPG score of 50.", () => rpgScore >=
50,
        () => (rpgScore / 50));
    theory.createAchievement(303, ac7,
"Wait, There's More?", "Reach an RPG
score of 75.", () => rpgScore >= 75,
        () => (rpgScore / 80));
    theory.createAchievement(304, ac7,
"Second Theory Cleared", "Reach an
RPG score of 100.", () => rpgScore >=
100,
        () => (rpgScore / 110));
    theory.createAchievement(305, ac7,
"Getting Stronger", "Reach an RPG
score of 160.", () => rpgScore >= 160,
        () => (rpgScore / 160));
    theory.createAchievement(306, ac7,
"Quarter of a Thousand", "Reach an
RPG score of 250.", () => rpgScore >=
250,
        () => (rpgScore / 250));
    theory.createAchievement(307, ac7,
"Level 35 Hitman", "Reach an RPG
score of 350.", () => rpgScore >= 350,
        () => (rpgScore / 350));
    theory.createAchievement(308, ac7,
"Overpowered", "Reach an RPG score
of 500.", () => rpgScore >= 500,
        () => (rpgScore / 500));
    theory.createAchievement(309, ac7,
"Really Big Numbers", "Reach an RPG
score of 750.", () => rpgScore >= 750,
        () => (rpgScore / 750));
    theory.createAchievement(310, ac7,
"Level 100 Boss", "Reach an RPG
score of 1000.", () => rpgScore >=
1000,
        () => (rpgScore / 1000));

    acs =
theory.createAchievementCategory(10
0, "Secrets");

theory.createSecretAchievement(1000
0, acs, "That's No Clicker", "Click the
screen rapidly and meaninglessly.",
"The opposite of idle",
        () => sc1Prog > 100);

theory.createSecretAchievement(1000
1, acs, "Visible During The Day",
"Make it so you have more of a higher
tier star than that of a lower tier star.",
"Reversed odds",
        () => stars[0] < stars[1] &&
stars[1] < stars[2] && stars[2] <
stars[3] && stars[3] < stars[4] &&
stars[4] < stars[5]);

theory.createSecretAchievement(1000
2, acs, "6×6", "Fully fill the Domino
Puzzle board.", "Entirely covered",
        () => !!(board && board.length ==
36 && board.findIndex(x => !x) < 0));

theory.createSecretAchievement(1000
3, acs, "Endurance", "Reach level
72000 on the \"↑ time by 0.1s\"
upgrade.", "Two hours",
        () => clicker.level > 71999);

theory.createSecretAchievement(1000
4, acs, "kaepsteeL", "Make the
product of c₅ to c₈ equals 3024000.",
"3024000",
        () => c5.level == 7 && c6.level ==
3 && c7.level == 3 && c8.level == 1);

    updateAvailability();

    theory.primaryEquationScale = 0.85;
}

var updateAvailability = () => {
    /////////////////
    // Normal Upgrades

    cTerms.isAvailable = prgSpeed.level
> 0;
    qTerms.isAvailable = prgSpeed.level
> 1;
    cur2Unlock.isAvailable =
prgSpeed.level > 2 && rTerms.level > 1
&& cTerms.level > 1 && qTerms.level >
3;
    gachaUnlock.isAvailable =
cur2Unlock.level > 0;
    gachaValue.isAvailable =
gachaUnlock.level > 0;
    gachaBulk.isAvailable =
gachaValue.level > 14;
    cur3Unlock.isAvailable =
gachaBulk.level > 73;
    prg3Speed.isAvailable =
qTerms3.isAvailable =
cur3Unlock.level > 0;
    diceUnlock.isAvailable =
prg3Speed.level > 2 && qTerms3.level
> 1;
    spUnlock.isAvailable =
prg3Speed.level > 5 && qTerms3.level
> 1;
    gachaValue2.isAvailable =
spUnlock.level > 0;
    rpgUnlock.isAvailable =
gachaValue2.level > 3;
    atkBoost.isAvailable =
workNerf.isAvailable = rpgUnlock.level
> 0;
    diceBoost.isAvailable =
diceBoost2.isAvailable =
atkBoost.level > 9 && workNerf.level >
9;

    b1.isAvailable = c1.isAvailable =
c2.isAvailable = stage == 0;
    c3.isAvailable = stage == 0 &&
cTerms.level > 0;
    c4.isAvailable = stage == 0 &&
cTerms.level > 1;

    b2.isAvailable = n1.isAvailable =
n2.isAvailable = stage == 1;



    c5.isAvailable = stage == 1 &&
skills[11].level > 0;
    c6.isAvailable = stage == 1 &&
skills[11].level > 1;
    c7.isAvailable = stage == 1 &&
skills[11].level > 2;
    c8.isAvailable = stage == 1 &&
skills[11].level > 3;

    c9.isAvailable = stage == 3 &&
skills[20].level > 0;
    c10.isAvailable = stage == 3 &&
skills[20].level > 1;
    c11.isAvailable = stage == 3 &&
skills[20].level > 2;
    c12.isAvailable = stage == 3 &&
skills[20].level > 3;


    clicker.isAvailable = stage == 2;
    minigame.isAvailable = stage == 2
&& skills[12].level > 0;
    perk.isAvailable = stage == 5 &&
diceMasteries ==
"############################
########";
    booster.isAvailable = stage == 5 &&
perks[11];

    for (skill of skills) skill.isAvailable =
stage == 2 && skill.level > 0;

    b3.isAvailable = n3.isAvailable =
n4.isAvailable = stage == 3;

    d4.isAvailable = d6.isAvailable =
d8.isAvailable =
    d10.isAvailable = d12.isAvailable =
d20.isAvailable =
    d4Master.isAvailable =
d6Master.isAvailable =
d8Master.isAvailable =
    d10Master.isAvailable =
d12Master.isAvailable =
d20Master.isAvailable = stage == 4;

    /////////////////////
    // Permanent Upgrades

    permPub.isAvailable =
permAll.isAvailable =
permAuto.isAvailable = stage == 0;
    perm1.isAvailable =
perm2.isAvailable = perm3.isAvailable
= stage == 1;
    perm4.isAvailable = stage == 1 &&
perm3.level > 0;

    perm5.isAvailable =
perm6.isAvailable = stage == 3;
    perm7.isAvailable = stage == 3 &&
cTerms.level > 0;
    perm8.isAvailable = stage == 3 &&
cTerms.level > 1;

    perm9.isAvailable =
perm10.isAvailable =
    perm11.isAvailable =
perm12.isAvailable = stage == 3;

    perm13.isAvailable =
perm14.isAvailable =
perm15.isAvailable = stage == 5;

    currency2.isAvailable =
cur2Unlock.level > 0;

    theory.primaryEquationHeight =
qTerms.level > 0 ? 75 : 50;
    theory.secondaryEquationHeight =
cur2Unlock.level > 0 ? 40 : 30;
}

var tick = (elapsedTime, multiplier) =>
{
    let dt =
BigNumber.from(elapsedTime *
multiplier);

    prg += dt *
getPrgSpeed(prgSpeed.level) *
getSkillEffect(0);
```

```
    if (prg >= 1) {
        let ptr =
Math.floor(Math.random() * (2 +
rTerms.level));

        let val = getC1(c1.level) *
getC2(c2.level);
        if (cTerms.level > 0) val *=
getC3(c3.level);
        if (cTerms.level > 1) val *=
getC4(c4.level);
        val = ((randSum(prg.floor()) *
getB1(b1.level - 1) + 1) * val.pow(1 /
(ptr + 1)));
        r[ptr] += val;

        if (qTerms.level > ptr) {
            q[ptr] = randSum(prg.floor()) *
getB1(b1.level - 1) + 1;
        }

        prg -= prg.floor();
        if (stage == 0) {

theory.invalidateQuaternaryValues();
        }
    }
    if (cur2Unlock.level > 0) prg2 += dt /
30 * getSkillEffect(1);
    if (prg2 >= 1) {
        let ptr =
Math.floor(Math.random() * (2 +
perm3.level + perm4.level) + 4);

        let val = (randSum(getN1(n1.level)
* getN2(n2.level) * prg2.floor()) *
(getB2(b2.level - 1) + 1));
        if (skills[11].level > 0) val *=
getC5(c5.level);
        if (skills[11].level > 1) val *=
getC6(c6.level);
```

```
        if (skills[11].level > 2) val *=
getC7(c7.level);
        if (skills[11].level > 3) val *=
getC8(c8.level);
        if (perm9.level > 0 && ptr == 4)
val *= q[4] + 1;
        if (perm10.level > 0 && ptr == 5)
val *= q[5] + 1;
        if (perm11.level > 0 && ptr == 6)
val *= q[6] + 1;
        if (perm12.level > 0 && ptr == 7)
val *= q[7] + 1;
        r[ptr] += val;

        prg2 -= prg2.floor();
    }
    if (gachaUnlock.level > 0) {
        let spd = 1 / 300 *
getSkillEffect(2);
        if (diceUnlock.level > 0) spd *=
((dicePoints + 1).log10() + 1).pow(1 /
getSkillEffect(17));
        prgGacha += dt * spd;
    }
    if (prgGacha >= 1) {
        gacha += prgGacha.floor();
        gachaTotal += prgGacha.floor();
        prgGacha -= prgGacha.floor();

theory.invalidatePrimaryEquation();

theory.invalidateSecondaryEquation();
    }
    if (cur3Unlock.level > 0) prg3 += dt
* getPrg3Speed(prg3Speed.level) *
getSkillEffect(13);
    if (prg3 >= 1) {
        let ptr =
Math.floor(Math.random() * (2 +
qTerms3.level) + 4);
```

```
        let val = 
(randSum(getN3(n3.level) * 
getN4(n4.level) * prg3.floor()) * 
getB3(b3.level));
        if (skills[20].level > 0) val *= 
getC9(c9.level);
        if (skills[20].level > 1) val *= 
getC10(c10.level);
        if (skills[20].level > 2) val *= 
getC11(c11.level);
        if (skills[20].level > 3) val *= 
getC12(c12.level);

        q[ptr] = val.max(q[ptr]);

        prg3 -= prg3.floor();

        if (stage == 3) {

theory.invalidateQuaternaryValues();
        }
    }
    if (diceUnlock.level > 0) prgDice += 
dt / 30 * getSkillEffect(14);
    if (prgDice >= 1) {

        let val = BigNumber.ZERO;
        let val2 = BigNumber.ZERO;
        for (let a = 0; a < 6; a++) {
            let d = [d4, d6, d8, d10, d12, 
d20][a].level;
            let v = [4, 6, 8, 10, 12, 20][a];
            if (d < 1) continue;
            let baseValue = 
BigNumber.from(Math.ceil(Math.rando
m() * v));
            let value = baseValue;
            if (diceMasteries[a * 6] == "#") 
value *= 10;
            if (diceMasteries[a * 6 + 1] == 
"#" && baseValue == v) {
```

```
                let ve =
Math.ceil(Math.random() * v);
                value *= ve;
                while (ve == v) {
                    ve =
Math.ceil(Math.random() * v);
                    value *= ve;
                }
            }
            if (diceMasteries[a * 6 + 2] ==
"#" && baseValue == 1) value *= v * v;
            if (diceMasteries[a * 6 + 3] ==
"#") value *= Math.ceil(Math.random()
* v);
            if (diceMasteries[a * 6 + 4] ==
"#" && Math.random() < .2) value *=
d;
            if (diceMasteries[a * 6 + 5] ==
"#") val2 += value;
            val += value * d;
        }

        val *= (prgDice * 1.1 **
diceBoost.level).floor();
        val2 *= (prgDice * 1.1 **
diceBoost2.level).floor();

        dicePoints += val;
        dicePoints2 += val2;

        prgDice -= prgDice.floor();

        if (stage == 4) {

theory.invalidatePrimaryEquation();

theory.invalidateQuaternaryValues();
        }
    }

    prgBar.progress = Math.min([prg,
prg2, prgGacha, prg3, prgDice,
ourHealth][stage].toNumber(), 1);

    if (stage == 1)
theory.invalidateQuaternaryValues();

    let bonus =
theory.publicationMultiplier;
    let pVal = dt * bonus * r[0] * r[1] *
getSkillEffect(3) * hiscoreEffs[0] *
getPerm5(perm5.level);
    if (rTerms.level > 0) pVal *= r[2];
    if (rTerms.level > 1) pVal *= r[3];
    if (qTerms.level > 0) pVal *= q[0];
    if (qTerms.level > 1) pVal *= q[1];
    if (qTerms.level > 2) pVal *= q[2];
    if (qTerms.level > 3) pVal *= q[3];
    if (perm2.level > 0) pVal *= r[5] + 1;
    if (perm13.level > 0) pVal *=
getPerm13(perm13.level);
    currency.value += pVal;

    if (cur2Unlock.level > 0) {
        let p2Val = (r[4] + r[5] + r[6] +
r[7]) * dt * getSkillEffect(4) *
hiscoreEffs[1] *
getPerm6(perm6.level);;
        if (perm1.level > 0) p2Val *= r[4] +
1;
        if (skills[8].level > 0) p2Val *=
bonus.pow(getSkillEffect(8) / 100);
        if (perm14.level > 0) p2Val *=
getPerm14(perm14.level);
        currency2.value += p2Val;
        let dil = getSkillEffect(9);
        dil = ((dil - 1) / dil);
        r[4] *= dil ** dt;
        r[5] *= dil ** dt;
        r[6] *= dil ** dt;
        r[7] *= dil ** dt;
    }
    minigame.isAutoBuyable =
clicker.isAutoBuyable =
perk.isAutoBuyable =
booster.isAutoBuyable = false;
    c4.maxLevel = Math.max(c4.level,
16 + getSkillEffect(10));

    if (cur3Unlock.level > 0) {
        let p3Val = (q[4] + q[5]);
        if (qTerms3.level > 0) p3Val +=
q[6];
        if (qTerms3.level > 1) {
            p3Val += q[7];
            p3Val *=
getPerm8(perm8.level);
        }
        if (diceUnlock.level > 0) {
            p3Val *= (dicePoints +
1).pow(1 / getSkillEffect(18));
        }
        if (perm15.level > 0) p3Val *=
getPerm15(perm15.level);
        if (skills[15].level > 0) p3Val *=
getSkillEffect(15);
        if (skills[19].level > 0) p3Val *=
bonus.pow(getSkillEffect(19) / 100);
        p3Val *= dt;
        currency3.value += p3Val;
    }

    if (dicePoints2 > 0) {
        dicePoints += dicePoints2 * dt;

theory.invalidatePrimaryEquation();
    }

    if (rpgUnlock.level > 0) {
        if(rpgBestLemma >
50*rpgTheory) {
            rpgBestLemma =
50*rpgTheory-1;
        }
        theirMaxHealth = (1e7 *
BigNumber.ONE.max(rpgLemma)) *
(.95 ** workNerf.level) *
(BigNumber.from(20).pow(rpgTheory
- 1));
        rpgScore = rpgBestLemma;
        if (rpgLemma > 0) {
            let seed = rpgTheory * 50 +
rpgLemma;
            seed = Math.floor((seed *
seed / 1000) % 9999991);
            let fact = seed % 4;

            let atkMult = BigNumber.ONE *
(1 + .15 * atkBoost.level);
            if (perks[11]) atkMult *= (1 +
pBoosters[fact] * (perks[31 + fact] ?
2 : 1));
            if (perks[21 + fact]) atkMult *=
2;
            if (perks[41]) atkMult *=
rpgTheory / 5 + 1;
            if (perks[42]) atkMult *=
rpgBestLemma / 25 + 1;

            ourHealth -= (theirMaxHealth *
theirHealth * dt) / dicePoints;
            theirHealth -= Math.min(1,
(dicePoints2 * ourHealth * dt) /
theirMaxHealth * atkMult);
            rpgBestLemma =
Math.max(rpgLemma -
theirHealth.toNumber() + 50 *
rpgTheory - 50, rpgBestLemma || 0);
            if (theirHealth <= 0) {
                theirHealth =
BigNumber.ONE;
                rpgLemma++;
                if (rpgLemma > 50) {
                    rpgTheory++;
                    rpgLemma = 0;
                    theirHealth =
BigNumber.ONE;
                }

theory.invalidatePrimaryEquation();

theory.invalidateSecondaryEquation();
            } else if (ourHealth <= 0) {
                ourHealth =
BigNumber.ZERO;
                theirHealth =
BigNumber.ONE;
                rpgLemma = 0;

theory.invalidatePrimaryEquation();

theory.invalidateSecondaryEquation();
            }

theory.invalidateQuaternaryValues();
        } else {
            ourHealth = (ourHealth + 1 /
(dicePoints + 1).pow(0.2) * dt).min(1);
        }
    }

    if (gameGrid && gameGrid.width > 0
&& gameGrid.heightRequest < 0)
        gameGrid.heightRequest =
Math.max(gameGrid.width, 0);
    if (invGrid && invGrid.width > 0 &&
invGrid.heightRequest < 0) {
        invGrid.heightRequest =
Math.max(invGrid.width / 3 - 10, 0);
        for (let a = 0; a < 3; a++)
invGrids[a].heightRequest =
Math.max(invGrid.width / 3, 0);
    }
}
```

```
var getInternalState = () => {
    let state = `${r[0]} ${r[1]} ${r[2]} $
{r[3]} ${q[0]} ${q[1]} ${q[2]} ${q[3]} $
{stage} ${r[4]} ${r[5]} ${r[6]} ${r[7]} $
{gacha} ${prgGacha} ` +
        `${stars[0]} ${stars[1]} $
{stars[2]} ${stars[3]} ${stars[4]} $
{stars[5]} ${hiscores[0]} $
{hiscores[1]} ${hiscores[2]} $
{hiscores[3]} ` +
        `${gachaTotal} ${q[4]} ${q[5]} $
{q[6]} ${q[7]} ${dicePoints} $
{dicePoints2} ${diceMasteries} $
{rpgTheory} ${rpgLemma} $
{ourHealth} ${theirHealth} ` +
        `${rpgBestLemma} ${perkPoints}
${perkPointsBought} ${pBoosters[0]}
${pBoosters[1]} ${pBoosters[2]} $
{pBoosters[3]} ${pBoostersBought}|$
{skillExp.join(" ")}|$
{Object.keys(perks).join(" ")}`;
    return state;

}

var setInternalState = (state) => {
    let values = state.split("|").map(x =>
x.split(" "));
    if (values.length > 0) {
        if (values[0].length > 0) r[0] =
parseBigNumber(values[0][0]);
        if (values[0].length > 1) r[1] =
parseBigNumber(values[0][1]);
        if (values[0].length > 2) r[2] =
parseBigNumber(values[0][2]);
        if (values[0].length > 3) r[3] =
parseBigNumber(values[0][3]);
        if (values[0].length > 4) q[0] =
parseBigNumber(values[0][4]);
        if (values[0].length > 5) q[1] =
parseBigNumber(values[0][5]);
        if (values[0].length > 6) q[2] =
parseBigNumber(values[0][6]);
        if (values[0].length > 7) q[3] =
parseBigNumber(values[0][7]);
        if (values[0].length > 8) stage =
+values[0][8];
        if (values[0].length > 9) r[4] =
parseBigNumber(values[0][9]);
        if (values[0].length > 10) r[5] =
parseBigNumber(values[0][10]);
        if (values[0].length > 11) r[6] =
parseBigNumber(values[0][11]);
        if (values[0].length > 12) r[7] =
parseBigNumber(values[0][12]);
        if (values[0].length > 13) gacha =
parseBigNumber(values[0][13]);
        if (values[0].length > 14)
prgGacha =
parseBigNumber(values[0][14]);
        if (values[0].length > 15) stars[0]
= +(values[0][15]);
        if (values[0].length > 16) stars[1]
= +(values[0][16]);
        if (values[0].length > 17) stars[2]
= +(values[0][17]);
        if (values[0].length > 18) stars[3]
= +(values[0][18]);
        if (values[0].length > 19) stars[4]
= +(values[0][19]);
        if (values[0].length > 20) stars[5]
= +(values[0][20]);
        if (values[0].length > 21)
hiscores[0] = +(values[0][21]);
        if (values[0].length > 22)
hiscores[1] = +(values[0][22]);
        if (values[0].length > 23)
hiscores[2] = +(values[0][23]);
        if (values[0].length > 24)
hiscores[3] = +(values[0][24]);
        if (values[0].length > 25)
```

```
gachaTotal =
parseBigNumber(values[0][25]);
        if (values[0].length > 26) q[4] =
parseBigNumber(values[0][26]);
        if (values[0].length > 27) q[5] =
parseBigNumber(values[0][27]);
        if (values[0].length > 28) q[6] =
parseBigNumber(values[0][28]);
        if (values[0].length > 29) q[7] =
parseBigNumber(values[0][29]);
        if (values[0].length > 30)
dicePoints =
parseBigNumber(values[0][30]);
        if (values[0].length > 31)
dicePoints2 =
parseBigNumber(values[0][31]);
        if (values[0].length > 32)
diceMasteries = values[0][32];
        if (values[0].length > 33)
rpgTheory = +(values[0][33]);
        if (values[0].length > 34)
rpgLemma = +(values[0][34]);
        if (values[0].length > 35)
ourHealth =
parseBigNumber(values[0][35]);
        if (values[0].length > 36)
theirHealth =
parseBigNumber(values[0][36]);
        if (values[0].length > 37)
rpgBestLemma = +(values[0][37]);
        if (values[0].length > 38)
perkPoints = +(values[0][38]);
        if (values[0].length > 39)
perkPointsBought = +(values[0][39]);
        if (values[0].length > 40)
pBoosters[0] = +(values[0][40]);
        if (values[0].length > 41)
pBoosters[1] = +(values[0][41]);
        if (values[0].length > 42)
pBoosters[2] = +(values[0][42]);
        if (values[0].length > 43)
```

```
pBoosters[3] = +(values[0][43]);
        if (values[0].length > 44)
pBoostersBought = +(values[0][44]);
    }
    if (values.length > 1) {
        for (let a = 0; a < values[1].length;
a++)
            skillExp[a] = +values[1][a];
    }
    if (values.length > 2) {
        for (let a = 0; a < values[2].length;
a++)
            perks[values[2][a]] = true;
    }

    updateScoreEffs();
}

var getPrimaryEquation = () => {
    if (stage == 0) {
        theory.primaryEquationScale =
.85;
        let pVal = "1, 2";
        if (rTerms.level > 0) pVal += ", 3";
        if (rTerms.level > 1) pVal += ", 4";

        let cVal = "c_1 c_2";
        if (cTerms.level > 0) cVal += "
c_3";
        if (cTerms.level > 1) cVal += "
c_4";

        let qVal = qTerms.level > 0 ? `\\\\
q_i \\leftarrow \\operatorname{rand}(1,
b_1) \\text{ if } \\exists \\text{ } q_i` : "";

        let result = `
            \\begin{cases}
                i \\in \\{${pVal}\\} \\\\
                r_i \\leftarrow r_i + \
\operatorname{rand}(1, b_1) \\sqrt[i]{$
{cVal}}
                ${qVal}
            \\end{cases}
         `;
        return result;
    } else if (stage == 1) {
        theory.primaryEquationScale =
.85;

        let pVal = "5, 6";
        if (perm3.level > 0) pVal += ", 7";
        if (perm4.level > 0) pVal += ", 8";

        let cVal = "";
        if (skills[11].level > 0) cVal +=
"c_5";
        if (skills[11].level > 1) cVal += "
c_6";
        if (skills[11].level > 2) cVal += "
c_7";
        if (skills[11].level > 3) cVal += "
c_8";

        let result = `
            \\begin{cases}
                i \\in \\{${pVal}\\} \\\\
                r_i \\leftarrow r_i + ${cVal}\
\displaystyle{\\sum^{n_1 n_2}_{0}}{\
\operatorname{rand}(1, b_2)}
            \\end{cases}
         `;
        return result;
    } else if (stage == 2) {
        theory.primaryEquationScale = 1;
        let result = `
            \\ominus = ${gacha.toString(0)}
         `;
        return result;
    } else if (stage == 3) {
        theory.primaryEquationScale =
.85;
```

```
        let pVal = "5, 6";
        if (qTerms3.level > 0) pVal += ",
7";
        if (qTerms3.level > 1) pVal += ",
8";

        let cVal = "";
        if (skills[20].level > 0) cVal +=
"c_9";
        if (skills[20].level > 1) cVal += "
c_{10}";
        if (skills[20].level > 2) cVal += "
c_{11}";
        if (skills[20].level > 3) cVal += "
c_{12}";

        let result = `
            \\begin{cases}
                i \\in \\{${pVal}\\} \\\\
                q_i \\leftarrow \\max(q_i, $
{cVal}\\displaystyle{\\sum^{n_3 n_4}
_{0}}{\\operatorname{rand}(0, b_3)})
            \\end{cases}
        `;
        return result;
    } else if (stage == 4) {
        theory.primaryEquationScale =
dicePoints2 > 0 ? .85 : 1;

        let result = `
            ${dicePoints2 > 0 ? `\\dot{\
\boxdot} = $
{dicePoints2.toNumber().toFixed(0)} \\\
\` : ""}
            \\boxdot = $
{dicePoints.toNumber().toFixed(0)} \\\\
            \\boxdot \\leftarrow \\boxdot + \
\sum d
        `;
        return result;
    } else if (stage == 5) {
        theory.primaryEquationScale = 1;
        let res = "\\begin{matrix}";
        res += `\\text{Theory #${rpgTheory}} \\\\ \\scriptstyle{\\text{-Probability Theory -}}`;
        res += "\\end{matrix}";

        return res;
    }
}

let seSeed = Math.floor(Math.random() * 2147483647);

var getSecondaryEquation = () => {

    let rVal = `\\dot{\\rho_1} = ${perm2.level > 0 ? "(r_6 + 1)" : ""} \\prod${cur2Unlock.level > 0 ? "^{4}_{i=1}" : "_{i}"}{r_i ${qTerms.level > 0 ? "q_i" : ""}}, \\quad ${theory.latexSymbol} = \\max \\rho_1`;
    if (stage == 1) rVal = `\\dot{\\rho_2} = ${perm1.level > 0 ? "(r_5 + 1)" : ""} \\sum^{8}_{i=5}{r_i}, \\quad \\dot{r_i} = \\frac{-r_i}{${getSkillEffect(9)}}`;
    else if (stage == 2) {
        extra_mult = 1;
        if (gachaBulk.level==74 && !(gacha/(1 + gachaBulk.level) < 100) && gacha != 0) {
            extra_mult = Math.pow(10, (Math.floor(Math.log10(gacha/(1 + gachaBulk.level)))-1));
        }
        if(gachaBulk.level<74 && !(gacha/(1 + gachaBulk.level) < 1000) && gacha != 0) {
            extra_mult = Math.pow(10,
(Math.floor(Math.log10(gacha/(1 +
gachaBulk.level)))-2));
        }
        let multi = Math.min(gacha, (1 +
gachaBulk.level)*extra_mult);
        if (multi == 0) rVal = "\
\text{Insufficient }\\ominus";
        else if (multi == 1) {
            let surpriseThings = [
                "open a loot box", "open a
gachapon", "open a surprise egg",
"draw one", "break a lucky block",
                "spin the wheel", "predict
the outcome", "play lottery", "roll a
die", "use a \\ominus", "summon
once",
                "play slots",
            ];
            rVal = `\\text{Tap to $
{surpriseThings[seSeed %
surpriseThings.length]}}`
        } else {
            let surpriseThings = [
                "open {mul} loot boxes",
"open {mul} gachapons", "open {mul}
surprise eggs", "draw {mul}",
                "break {mul} lucky blocks",
"roll {mul} dice", "use {mul} \\ominus",
"summon {mul}",
            ];
            rVal = `\\text{Tap to $
{surpriseThings[seSeed %
surpriseThings.length].replace("{mul}"
, multi)}}`
        }
    }
    else if (stage == 3) {
        rVal = `\\dot{\\rho_3} = \\sum^{8}
_{i=5}{q_i}`;
        if (diceUnlock.level > 0) {
            let rt = getSkillEffect(18)
            rVal += rt > 1 ? "\\sqrt[" + (rt ==
2 ? "" : rt) + "]{\\boxdot + 1}" : "(\
\boxdot + 1)^{" + (rt < 1 ? (1 /
rt).toFixed(2) : 1) + "}";
        }
    }
    else if (stage == 4) {
        rVal = `\\dot{\\ominus} \\leftarrow
\\dot{\\ominus} \\times \\sqrt[$
{getSkillEffect(17)}]{\\log(\\boxdot + 1)
+ 1}`;
    }
    else if (stage == 5) {
        let seed = rpgTheory * 50 +
rpgLemma;
        seed = Math.floor((seed * seed /
1000) % 9999991);
        let fact = seed % 4;

        let res = "\\begin{matrix}";
        res += rpgLemma ? `\\text{Lemma
${rpgLemma} / 50} \\\\ \\scriptstyle{\
\text{` + (perks[11] ?
            ["Data Collecting...",
"Analyzing...", "Evaluating...",
"Revising..."][fact] : `Researching...`)
+ `}}` : `\\text{Tap to begin research}`;
        res += "\\end{matrix}";

        return res;
    }

    return `${rVal}`;
}
var getQuaternaryEntries = () => {
    quaternaryEntries = [];

    if (stage == 0) {
        quaternaryEntries.push(new
QuaternaryEntry("r_1",
r[0].toString()));
```

```
        quaternaryEntries.push(new
QuaternaryEntry("r_2",
r[1].toString()));
        quaternaryEntries.push(new
QuaternaryEntry("r_3", rTerms.level >
0 ? r[2].toString() : null));
        quaternaryEntries.push(new
QuaternaryEntry("r_4", rTerms.level >
1 ? r[3].toString() : null));

        if (qTerms.level > 0) {
            quaternaryEntries.push(new
QuaternaryEntry("q_1",
q[0].toString()));
            quaternaryEntries.push(new
QuaternaryEntry("q_2", qTerms.level >
1 ? q[1].toString() : null));
            quaternaryEntries.push(new
QuaternaryEntry("q_3", qTerms.level >
2 ? q[2].toString() : null));
            quaternaryEntries.push(new
QuaternaryEntry("q_4", qTerms.level >
3 ? q[3].toString() : null));
        }
    } else if (stage == 1) {
        quaternaryEntries.push(new
QuaternaryEntry("r_5",
r[4].toString()));
        quaternaryEntries.push(new
QuaternaryEntry("r_6",
r[5].toString()));
        quaternaryEntries.push(new
QuaternaryEntry("r_7", perm3.level >
0 ? r[6].toString() : null));
        quaternaryEntries.push(new
QuaternaryEntry("r_8", perm4.level >
0 ? r[7].toString() : null));
    } else if (stage == 2) {
        quaternaryEntries.push(new
QuaternaryEntry("\\bigstar_1",
stars[0].toString()));
```

```
        quaternaryEntries.push(new
QuaternaryEntry("\\bigstar_2",
stars[1].toString()));
        quaternaryEntries.push(new
QuaternaryEntry("\\bigstar_3",
stars[2].toString()));
        quaternaryEntries.push(new
QuaternaryEntry("\\bigstar_4",
stars[3].toString()));
        quaternaryEntries.push(new
QuaternaryEntry("\\bigstar_5",
stars[4].toString()));
        quaternaryEntries.push(new
QuaternaryEntry("\\bigstar_6",
stars[5].toString()));
    } else if (stage == 3) {
        quaternaryEntries.push(new
QuaternaryEntry("q_5",
q[4].toString()));
        quaternaryEntries.push(new
QuaternaryEntry("q_6",
q[5].toString()));
        quaternaryEntries.push(new
QuaternaryEntry("q_7", qTerms3.level
> 0 ? q[6].toString() : null));
        quaternaryEntries.push(new
QuaternaryEntry("q_8", qTerms3.level
> 1 ? q[7].toString() : null));
    } else if (stage == 5) {
        quaternaryEntries.push(new
QuaternaryEntry(rpgTheory > 3 ?
rpgTheory - 3 : "", rpgTheory > 3 ?
50 : ""));
        quaternaryEntries.push(new
QuaternaryEntry(rpgTheory > 2 ?
rpgTheory - 2 : "", rpgTheory > 2 ?
50 : ""));
        quaternaryEntries.push(new
QuaternaryEntry(rpgTheory > 1 ?
rpgTheory - 1 : "", rpgTheory > 1 ? 50 :
""));
```

```
        quaternaryEntries.push(new
QuaternaryEntry(rpgTheory,
(rpgBestLemma - rpgTheory * 50 +
50).toFixed(2) + " / 50"));
        quaternaryEntries.push(new
QuaternaryEntry(rpgTheory + 1, null));
        quaternaryEntries.push(new
QuaternaryEntry(rpgTheory + 2, null));
        quaternaryEntries.push(new
QuaternaryEntry(rpgTheory + 3, null));
    } else {
        return [];
    }

    return quaternaryEntries;
}

var postPublish = () => {
    r = [BigNumber.ONE,
BigNumber.ONE, BigNumber.ONE,
BigNumber.ONE,
        BigNumber.ZERO,
BigNumber.ZERO, BigNumber.ZERO,
BigNumber.ZERO];
    q = [BigNumber.ONE,
BigNumber.ONE, BigNumber.ONE,
BigNumber.ONE,
        BigNumber.ZERO,
BigNumber.ZERO, BigNumber.ZERO,
BigNumber.ZERO];

    d4.level = d4Master.level;
    d6.level = d6Master.level;
    d8.level = d8Master.level;
    d10.level = d10Master.level;
    d12.level = d12Master.level;
    d20.level = d20Master.level;



    theory.invalidateQuaternaryValues();
}
```

```
let tSkill;

var addSkillExp = (id, value) => {
    skillExp[id] += value;
    let add = 0
    let cost =
skillData[id].starCost.getCost(skills[id]
.level).toNumber();
    while (skillExp[id] >= cost) {
        if (skills[id].maxLevel &&
skills[id].level + add >=
skills[id].maxLevel) {
            skillExp[id] = 0;
            break;
        }
        skillExp[id] -= cost;
        add += 1;
        cost =
skillData[id].starCost.getCost(skills[id]
.level + add).toNumber();
    }
    if (add > 0) {
        tSkill = id;
        skills[id].level += add;
    }
}

var getEquationOverlay = () =>
ui.createGrid({
    margin: new Thickness(40, 0, 40,
0),
    children: [
        prgBar =
ui.createProgressBar({ progress: 0,
verticalOptions:
LayoutOptions.START }),

        ui.createGrid({
            opacity: () => stage == 5 ? 1 :
0,
            children: [
                ui.createGrid({
                    columnDefinitions: ["7*",
"3*"],
                    children: [

ui.createLatexLabel({ text: "Power",
                            margin: new
Thickness(0, 10), horizontalOptions:
LayoutOptions.END, fontSize: 10
                        }),
                        ui.createLatexLabel({
                            text: () => (dicePoints
* ourHealth) + " / " + dicePoints,
                            fontSize: 10,
                            margin: new
Thickness(0, 10),
                        }),

ui.createLatexLabel({ text: "Work",
                            margin: new
Thickness(0, 10), horizontalOptions:
LayoutOptions.END, verticalOptions:
LayoutOptions.END, fontSize: 10
                        }),
                        ui.createLatexLabel({
                            text: () =>
(theirMaxHealth * theirHealth) + " / " +
theirMaxHealth,
                            fontSize: 10,
                            verticalOptions:
LayoutOptions.END,
                            margin: () =>
dicePoints < 1e12 ? new Thickness(0,
7) : new Thickness(0, 10),
                        }),
                    ],
                }),
                ui.createProgressBar({
                    progress: () =>
theirHealth.toNumber(),
verticalOptions: LayoutOptions.END,
            }),
          ],
        }),
    ],
    onTouched: (e) => {
        if (e.type != TouchType.PRESSED)
return;
        if (stage == 2) {
            if (gacha < 1) return;
            extra_mult = 1;
            if (gachaBulk.level==74 && !
(gacha/(1 + gachaBulk.level) < 100) &&
gacha != 0) {
                extra_mult = Math.pow(10,
(Math.floor(Math.log10(gacha/(1 +
gachaBulk.level)))-1));
            }
            if(gachaBulk.level<74 && !
(gacha/(1 + gachaBulk.level) < 1000)
&& gacha != 0) {
                extra_mult = Math.pow(10,
(Math.floor(Math.log10(gacha/(1 +
gachaBulk.level)))-2));
            }
            let multi = Math.min(gacha, 1 +
gachaBulk.level);
            let odds = [5, 4, 3, 2, 1,
0].map((x) => Math.pow(5 -
gachaValue.level * .2, x));
            for (let n = 0; n < multi; n++) {
                let osum = odds.reduce((x,
y) => x + y);
                for (let a = 0; a < 6; a++) {
                    let rand = Math.random()
< odds[a] / osum ? 1 : 0;
                    stars[a] += rand *
(gachaValue2.level + 1)*extra_mult;
                    if (rand > 0) break;
                    osum -= odds[a];
                }
                tSkill = 0;
                let a = 0;
                while (a < 1000 || tSkill >=
13) {
                    tSkill =
Math.floor(Math.random() *
skillData.length);
                    if (tSkill < 8) break;
                    else if (tSkill < 13 &&
Math.random() < 0.5) break;
                    else if (spUnlock.level > 0
&& Math.random() < 0.02) break;
                    a++;
                }
                let skill = skills[tSkill];
                if (skill.level == 0) skill.level
+= 1;
                tSkill = null;
            }
            gacha -= multi*extra_mult;
            seSeed =
Math.floor(Math.random() *
2147483647);

theory.invalidatePrimaryEquation();

theory.invalidateSecondaryEquation();

theory.invalidateQuaternaryValues();
            updateAvailability();
        } else if (stage == 5 &&
rpgLemma == 0) {
            rpgLemma = 1;

theory.invalidateSecondaryEquation();
        } else sc1Prog++;
    }
});

var isCurrencyVisible = (index) =>
index < 1 ||
    (index == 1 && cur2Unlock.level >
0) || (index == 2 && cur3Unlock.level >
0);
var getPublicationMultiplier = (tau) =>
{
    let mul = (BigNumber.E.pow((tau +
1).log().pow(.724)) / 500 *
getSkillEffect(5));
    if (qTerms3.level > 0) mul *=
getPerm7(perm7.level);
    return mul;
}
var getPublicationMultiplierFormula =
(symbol) => {
    let mul = getSkillEffect(5);
    if (qTerms3.level > 0) mul *=
getPerm7(perm7.level);
    return "\\frac{e^{\\ln(" + symbol + "
+ 1)^{0.724}}}{500}" + (mul > 1 ? "\
\times" + mul.toString(0) : "");
}
var getTau = () => currency.value;
var getCompletion = () => {
    let count = 0;
    for (let ach of theory.achievements)
{
        if (ach.isUnlocked) count++;
    }
    return Math.floor(count /
theory.achievements.length * 100);
};
var get2DGraphValue = () => {
    if (stage > 4) return 0;
    let cur = [currency.value,
currency2.value, currency2.value,
currency3.value, currency3.value]
[stage];
    return cur.sign * (BigNumber.ONE +
cur.abs()).log10().toNumber();
}
var get3DGraphPoint = () => {
    if (stage > 4) return new
Vector3((ourHealth).toNumber() * 2 -
1, (theirHealth).toNumber() - 0.5, 0);
    return new Vector3(0, 0, 0);
}

var getPrgSpeed = (level) => 0.1 * (2
** level);
var getPrg3Speed = (level) => 0.02 *
(1.5 ** level);
var getB1 = (level) =>
BigNumber.from(level +
2).pow(getSkillEffect(6)) *
(perm4.level > 0 ? r[7] + 1 : 1);
var getB2 = (level) =>
BigNumber.from(level +
2).pow(getSkillEffect(7));
var getB3 = (level) =>
BigNumber.from(level +
1).pow(getSkillEffect(16));

var getC1 = (level) =>
Utils.getStepwisePowerSum(level, 2,
10, 1) * (perm3.level > 0 ? r[6] + 1 : 1);
var getC2 = (level) =>
BigNumber.TWO.pow(level);
var getC3 = (level) =>
Utils.getStepwisePowerSum(level, 2,
10, 1);
var getC4 = (level) =>
BigNumber.TWO.pow(level);
var getC5 = (level) =>
BigNumber.TWO.pow(level);
var getC6 = (level) =>
BigNumber.THREE.pow(level);
var getC7 = (level) =>
BigNumber.FIVE.pow(level);
var getC8 = (level) =>
BigNumber.SEVEN.pow(level);
var getC9 = (level) =>
BigNumber.from(level + 1).pow(2);
```

```
var getC10 = (level) =>
BigNumber.from(level + 1).pow(3);
var getC11 = (level) =>
BigNumber.from(level + 1).pow(4);
var getC12 = (level) =>
BigNumber.from(level + 1).pow(5);

var getN1 = (level) =>
Utils.getStepwisePowerSum(level, 2,
10, 1);
var getN2 = (level) =>
BigNumber.TWO.pow(level);
var getN3 = (level) =>
Utils.getStepwisePowerSum(level, 2,
10, 1);
var getN4 = (level) =>
BigNumber.TWO.pow(level);

var getPerm5 = (level) =>
BigNumber.HUNDRED.pow(level);
var getPerm6 = (level) =>
BigNumber.FIVE.pow(level);
var getPerm7 = (level) =>
BigNumber.from(20).pow(level);
var getPerm8 = (level) =>
BigNumber.TWO.pow(level);

var getPerm13 = (level) =>
BigNumber.from(level +
1).pow(rpgScore);
var getPerm14 = (level) =>
BigNumber.from(level +
1).pow(rpgScore / 2);
var getPerm15 = (level) =>
BigNumber.from(level +
1).pow(rpgScore / 4);

var canGoToPreviousStage = () =>
stage > 0;
var goToPreviousStage = () => {
    stage -= 1;
    theory.clearGraph();
    theory.invalidatePrimaryEquation();

theory.invalidateSecondaryEquation();
    theory.invalidateQuaternaryValues();
    updateAvailability();
};
var canGoToNextStage = () => (stage
< 1 && cur2Unlock.level > 0) ||
    (stage < 2 && gachaUnlock.level >
0) || (stage < 3 && cur3Unlock.level >
0) ||
    (stage < 4 && diceUnlock.level > 0)
|| (stage < 5 && rpgUnlock.level > 0);
var goToNextStage = () => {
    stage += 1;
    theory.clearGraph();
    theory.invalidatePrimaryEquation();

theory.invalidateSecondaryEquation();
    theory.invalidateQuaternaryValues();
    updateAvailability();
};

var randSum = (itr) => {
    let num = BigNumber.ZERO;
    for (let a = 0; a < Math.min(itr, 100);
a++) {
        num += Math.random();
    }
    let mul = itr / 100;
    if (mul < 1) mul = 1;
    return num * mul;
}

var isPopupOpen = false;

var skillBulk = 1;

var showSkillPopup = (id) => {
    let skill = skillData[id];
    let skillInfoText =
ui.createLatexLabel({ text:
skills[id].getInfo(),
horizontalTextAlignment:
TextAlignment.CENTER, margin: new
Thickness(0, 10, 0, 6) });

    let levelText =
ui.createLatexLabel({text: "Level: " +
skills[id].level, row: 0, column: 0});
    let expText =
ui.createLatexLabel({text:
(skillExp[id].toFixed(0) + " / " +
skill.starCost.getCost(skills[id].level).t
oString(0)),
        horizontalOptions:
LayoutOptions.END_AND_EXPAND,
horizontalTextAlignment:
TextAlignment.END, row: 0, column:
2 });
    let expBar =
ui.createProgressBar({ progress:
Math.min(skillExp[id] /
skill.starCost.getCost(skills[id].level).t
oNumber(), 1) });

    let bulkText = ui.createLatexLabel({
        text: Number.isFinite(skillBulk) ?
Localization.get("BuyablesCostN",
skillBulk) :
Localization.get("BuyablesCostMax"),
        horizontalOptions:
LayoutOptions.END_AND_EXPAND,
horizontalTextAlignment:
TextAlignment.END,
    });

    let btns = [null, null, null, null, null,
null];
    for (let a = 0; a < 6; a++) {
        btns[a] = ui.createButton({
            text: "★" + "₁₂₃₄₅₆"[a] + " → +" + skill.starValue[a],
            row: Math.floor(a / 3),
            column: a % 3,
            opacity: stars[a] >= 1 ? 1 : 0.5,
            inputTransparent: stars[a] >= 1 ? false : true,
            onClicked: () => {
                if (stars[a] < 1) return;
                let max = skillData[id].maxLevel ? Math.ceil((skill.starCost.getSum(skills[id].level, skill.maxLevel) - skillExp[id]) / skill.starValue[a]) : Infinity;
                max = Math.min(stars[a], max + 1, skillBulk);
                addSkillExp(id, skill.starValue[a] * max);
                stars[a] -= max;
                btns[a].opacity = stars[a] >= 1 ? 1 : 0.5;
                btns[a].inputTransparent = stars[a] >= 1 ? false : true;
                skillInfoText.text = skills[id].getInfo();
                levelText.text = "Level: " + skills[id].level;
                expText.text = (skillExp[id].toFixed(0) + " / " + skill.starCost.getCost(skills[id].level).toString(0));
                expBar.progress = skillExp[id] / skill.starCost.getCost(skills[id].level).toNumber();

                if (skills[id].maxLevel && skills[id].level >= skills[id].maxLevel)
                    popup.hide();
```

```
theory.invalidateQuaternaryValues();
            }
        })
    }

    let popup = ui.createPopup({
        isPeekable: true,
        onDisappearing: () =>
isPopupOpen = false,
        content: ui.createStackLayout({
            children: [
                ui.createLatexLabel({ text: "\
\textbf{" + skill.name() + "}", fontSize:
17, horizontalTextAlignment:
TextAlignment.CENTER }),
                skillInfoText,
                ui.createLatexLabel({ text:
"(" + skill.effectText(skills[id].level) +
")", fontSize: 11, opacity: 0.65,
horizontalTextAlignment:
TextAlignment.CENTER }),

ui.createBox({ heightRequest: 1,
margin: new Thickness(0, 8) }),
                expBar,
                ui.createGrid({
                    columnDefinitions: ["1*",
"auto", "1*"],
                    children: [
                        levelText, expText,
                    ]
                }),

ui.createBox({ heightRequest: 1,
margin: new Thickness(0, 8) }),
                ui.createGrid({
                    children: [
                        bulkText,
                    ],
                    onTouched: (e) => {
                        log(e.type);
                        if (e.type ==
TouchType.PRESSED) {
                            let modes = [1, 10, 25,
100, Infinity];
                            skillBulk =
modes[(modes.indexOf(skillBulk) + 1)
% modes.length];
                            bulkText.text =
Number.isFinite(skillBulk) ?
Localization.get("BuyablesCostN",
skillBulk) :
Localization.get("BuyablesCostMax");
                        }
                    }
                }),
                ui.createGrid({
                    columnDefinitions: ["1*",
"1*", "1*"],
                    rowDefinitions: ["1*", "1*"],
                    children: btns,
                    margin: new Thickness(0,
5, 0, 0),
                }),

ui.createBox({ heightRequest: 1,
margin: new Thickness(0, 8) }),
                ui.createButton({ text:
Localization.get("AutoPrestigeClose"),
onClicked: () => { popup.hide() } })
            ]
        })
    });
    popup.show();
    isPopupOpen = true;
}

var showMinigamePopup = () => {
    let popup = ui.createPopup({
        title: "Minigames",
        onDisappearing: () =>
isPopupOpen = false,
        content: ui.createStackLayout({
            children: [
                ui.createLatexLabel({
                    text: "Play minigames to
skip $\\ominus$ time. \\\\ Additionally,
you gain bonuses based on your best
score in each minigame.",
                    horizontalTextAlignment:
TextAlignment.CENTER, margin: new
Thickness(0, 0, 0, 8)
                }),
                ui.createButton({ text: "Play
Block Puzzle\n" +
                    "Best: " + hiscores[0] + "
→ " + hiscoreEffs[0] + "× ρ₁ gain",
heightRequest: 60, onClicked: () => {
                    showBlockPuzzlePopup();
                    popup.hide();
                } }),
                skills[12].level > 1 ?
ui.createButton({ text: "Play Domino
Puzzle\n" +
                    "Best: " + hiscores[1] + "
→ " + hiscoreEffs[1] + "× ρ₂ gain",
heightRequest: 60, onClicked: () => {

showDominoPuzzlePopup();
                    popup.hide();
                } }) : ui.createButton({ text:
"Locked", heightRequest: 60, opacity:
0.5, inputTransparent: true }),
                ui.createButton({ text: "New
minigames coming soon...", opacity:
0.5, inputTransparent: true }),

ui.createBox({ heightRequest: 1,
margin: new Thickness(0, 8) }),
                ui.createButton({ text:
Localization.get("AutoPrestigeClose"),
onClicked: () => { popup.hide() } }),
            ]
        })
    });
    popup.show();
    isPopupOpen = true;
}

var gameScore = 0;
var gameStreak = 0;
var scoreLabel = null;
var hiscoreLabel = null;

var board = null;
var gameGrid = null;
var invGrid = null;
var invGrids = null;
var invs = null;
var invSel = null;
var invGhost = null;
var endButton = null;

var updateScoreEffs = () => {
    hiscoreEffs[0] =
BigNumber.from(hiscores[0] +
10).log10().pow(16);
    hiscoreEffs[1] =
BigNumber.from(hiscores[1] +
10).log10().pow(2);
    hiscoreEffs[2] =
BigNumber.from(hiscores[2] +
10).log10().pow(5);
}

var getMinigameColor = (val) => {
    if (val == 0) return
Color.MINIGAME_TILE_DARK;
    if (val == 1) return
Color.MINIGAME_TILE_LIGHT;
    let ret = Color.fromRgb(val * .2, val
* .2, val * .2);
```

```
    return ret;
}

var getInvSize = (inv) => {
    let w = inv.length;
    let h = inv[0].length;

    let m = [0, 0];

    for (let x = 0; x < w; x++) for (let y =
0; y < h; y++) {
        if (inv[x][y] != " ") {
            m = [Math.max(m[0], x),
Math.max(m[1], y)];
        }
    }

    return m;
}

var spinInv = (inv) => {
    let w = inv.length;
    let h = inv[0].length;

    let i = Array(h).fill("");
    for (let x = 0; x < w; x++) for (let y =
0; y < h; y++) {
        i[y] += inv[x][h - y - 1];
    }

    return i;
}

var canPlaceInv = (inv, pos) => {

    let size = Math.sqrt(board.length);
    if (pos[0] < 0 || pos[1] < 0 || pos[0]
>= size - inv.size[0] || pos[1] >= size -
inv.size[1]) return false;

    for (let x = 0; x <= inv.size[0]; x++)
for (let y = 0; y <= inv.size[1]; y++) {
        let p = (pos[0] + x) * size + pos[1]
+ y;
        if ((inv[x][y] && inv[x][y] != " ")
&& board[p]) return false;
    }

    return true;
}

var canContinue = () => {
    let size = Math.sqrt(board.length);
    for (let a = 0; a < 3; a++) {
        if (!invs[a] || !invs[a].size)
continue;
        for (let x = 0; x < size -
invs[a].size[0]; x++) for (let y = 0; y <
size - invs[a].size[1]; y++) {
            if (canPlaceInv(invs[a], [x, y]))
return true;
        }
    }
    return false;
}

var getBoardNum = (row, col) => {
    let size = Math.sqrt(board.length);
    return row * size + col;
}

////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////
///////
// Block Puzzle

var showBlockPuzzlePopup = () => {
    if (gameGrid) return;

    board = Array(81).fill(false);
    blocks = [
        ["#"],
        ["#", "#"],
        ["#", "#", "#"],
        ["#", "#", "#", "#"],
        ["# ", "# ", "##"],
        [" #", " #", "##"],
        [" #", "##", "# "],
        ["# ", "##", " #"],
        ["##", "##"],
        [" # ", "###", " # "],
        ["###", "#  ", "#  "],
        ["# ", "##"],
        ["###", " # "],
        ["###", "# #"],
    ]

    gameGrid = ui.createGrid({
        columnSpacing: 0,
        rowSpacing: 0,
    });
    let c = [];
    for (let x = 0; x < 9; x++) for (let y =
0; y < 9; y++) c.push(ui.createFrame({
        column: x,
        row: y,
        backgroundColor:
getMinigameColor((x % 6 < 3) ^ (y %
6 < 3) ? 0.5 : 0),
        borderColor:
Color.MINIGAME_TILE_BORDER,
        cornerRadius: 4,
    }));
    gameGrid.children = c;

    invGrids = [ui.createGrid(),
ui.createGrid(), ui.createGrid()];
    invs = [0, 0, 0].map((x) => {
        let b =
blocks[Math.floor(Math.random() *
blocks.length)];
        let s = Math.random() * 3;
        for (let a = 0; a < s; a++) b =
spinInv(b);
        b.size = getInvSize(b);
        return b;
    });
    for (let a = 0; a < 3; a++) {
        invGrids[a].columnDefinitions =
invGrids[a].rowDefinitions = ["1*", "1*",
"1*", "1*"];
        invGrids[a].columnSpacing =
invGrids[a].rowSpacing = 0;
        invGrids[a].column = a;
        let t;
        let c = [];
        for (let x = 0; x < 4; x++) for (let y
= 0; y < 4; y++)
c.push(ui.createFrame({
            column: x,
            row: y,
            heightRequest: 50,
            borderColor:
Color.MINIGAME_TILE_BORDER,
            backgroundColor:
Color.MINIGAME_TILE_LIGHT,
            cornerRadius: 4,
            opacity: invs[a][x] ? (invs[a][x]
[y] == "#" ? 1 : 0) : 0
        }));
        invGrids[a].onTouched = (e) => {
            if (invSel && invSel !=
invGrids[a]) return;
            if (!invs[a]) return;

            let t = invGrid.width / 9;
            let pos = [
                e.absoluteX + invGrids[a].x -
t / 2 * (invs[a].size[0] + 1),
                e.absoluteY + invGrid.y - t *
(invs[a].size[1] + 2),
            ];
            let tPos = [
                Math.round(pos[0] / t),
                Math.round((pos[1] -
gameGrid.y) / t),
            ];

            if (e.type.isReleased()) {
                invSel = null;
                let arr =
Array.from(invGhost.children);
                invGhost.children = [];
                invGrids[a].children = arr;

                if (canPlaceInv(invs[a],
tPos)) {
                    for (let x = 0; x <=
invs[a].size[0]; x++) for (let y = 0; y <=
invs[a].size[1]; y++) {
                        let p = (tPos[0] + x) * 9
+ tPos[1] + y;
                        if (invs[a][x][y] != " ") {
                            board[p] = true;

gameGrid.children[p].backgroundColo
r = getMinigameColor(1);
                            gameScore += 1;
                        }
                    }
                    invs[a] = null;
                    for (let b of
invGrids[a].children) b.opacity = 0;

                    if (invs[0] == null &&
invs[1] == null && invs[2] == null) {
                        invs = [0, 0, 0].map((x)
=> {
                            let b =
blocks[Math.floor(Math.random() *
blocks.length)];
                            let s = Math.random()
* 3;
                            for (let a = 0; a < s; a+
+) b = spinInv(b);
```

```
                        b.size =
getInvSize(b);
                        return b;
                    });
                    for (let b = 0; b < 3; b+
+) {
                        for (let t of
invGrids[b].children)
                            t.opacity = invs[b]
[t.column] ? (invs[b][t.column][t.row]
== "#" ? 1 : 0) : 0;
                    }
                }

                let del =
Array(81).fill(false);
                let delNum = 0;

                for (let x = 0; x < 9; x++) {
                    let row = true;
                    let col = true;
                    let blk = true;
                    for (let y = 0; y < 9; y++)
{
                        if (!
board[getBoardNum(x, y)]) row =
false;
                        if (!
board[getBoardNum(y, x)]) col = false;
                        if (!
board[getBoardNum((x % 3) * 3 + (y
% 3), Math.floor(x / 3) * 3 +
Math.floor(y / 3))]) blk = false;
                    }
                    if (row) delNum++;
                    if (col) delNum++;
                    if (blk) delNum++;
                    for (let y = 0; y < 9; y++)
{
                        if (row)
del[getBoardNum(x, y)] = true;
```

```
                            if (col)
del[getBoardNum(y, x)] = true;
                            if (blk)
del[getBoardNum((x % 3) * 3 + (y %
3), Math.floor(x / 3) * 3 +
Math.floor(y / 3))] = true;
                        }
                    }
                    if (delNum) {
                        let t = 0;
                        for (let p = 0; p < 81; p+
+) if (del[p]) {
                            board[p] = false;

gameGrid.children[p].backgroundColo
r = getMinigameColor(((p % 9) % 6 <
3) ^ (p % 54 < 27) ? 0.5 : 0);
                            t++;
                        }
                        gameStreak++;
                        gameScore += 1 + t *
delNum * gameStreak;
                    } else gameStreak = 0;

                    if (!canContinue()) {
                        gameGrid.opacity =
0.75;
                        invGrid.opacity = 0;
                        invGrid.heightRequest
-= 50;

endButton.heightRequest = 50;
                        endButton.text = "Gain "
+ Math.floor(gameScore ** 1.1) + " \"↑
time by 0.1s\" clicks";

                        hiscores[0] =
Math.max(hiscores[0], gameScore);
                        hiscoreLabel.text =
hiscores[0].toString();
                    }
```

```
                scoreLabel.text =
gameScore.toString();
            }
        } else if (e.type ==
TouchType.PRESSED ||  e.type ==
TouchType.MOVED) {
            if (!invSel) {
                let arr =
Array.from(invGrids[a].children);
                invGrids[a].children = [];
                invGhost.children = arr;
                invGhost.widthRequest =
invGhost.heightRequest =
invGrid.width / 9 * 4 - 8;
            }

            invSel = invGrids[a];
            invGhost.translationX =
pos[0];
            invGhost.translationY =
pos[1];
        }
    }
    invGrids[a].children = c;
};

let popup = ui.createPopup({
    title: "Block Puzzle",
    onDisappearing: () => {
        clicker.level +=
Math.floor(gameScore ** 1.1);
        hiscores[0] =
Math.max(hiscores[0], gameScore);
        updateScoreEffs();
        gameScore = 0;
        board = gameGrid = invs =
invGrid = invGrids = invSel =
scoreLabel = hiscoreLabel = null;
    },
    content: ui.createGrid({
        children: [
            ui.createStackLayout({
                children: [

ui.createLatexLabel({ text: "Place
blocks to fill lines or 3x3 squares.",
horizontalTextAlignment:
TextAlignment.CENTER, margin: new
Thickness(0, 10, 0, 6) }),
                    ui.createGrid({
                        margin: new
Thickness(0, 0, 0, 6),
                        columnDefinitions:
["1*", "1*"],
                        rowDefinitions: ["28",
"28"],
                        children: [

ui.createLatexLabel({ text: "Score: ",
horizontalOptions:
LayoutOptions.END, row: 0, column: 0,
fontSize: 17, }),
                            scoreLabel =
ui.createLatexLabel({ text: "0", row: 0,
column: 1, fontSize: 17, }),

ui.createLatexLabel({ text: "Best
Score: ", horizontalOptions:
LayoutOptions.END, row: 1, column: 0,
fontSize: 17, }),
                            hiscoreLabel =
ui.createLatexLabel({ text:
hiscores[0].toString(), row: 1, column:
1, fontSize: 17, }),
                        ],
                    }),
                    gameGrid,
                    invGrid = ui.createGrid({
                        margin: new
Thickness(0, 6, 0, 0),
                        columnDefinitions:
```

```
["1*", "1*", "1*"],
                              children: invGrids,
                          }),
                          endButton =
ui.createButton({
                              text: "Replay",
                              heightRequest: 0,
                              onClicked: () => {
                                  clicker.level +=
Math.floor(gameScore ** 1.1);
                                  gameScore = 0;
                                  scoreLabel.text =
gameScore.toString();

                                  gameGrid.opacity =
1;
                                  invGrid.opacity = 1;

invGrid.heightRequest += 50;

endButton.heightRequest = 0;

                                  for (let p = 0; p <
81; p++) {
                                      board[p] = false;

gameGrid.children[p].backgroundColo
r = getMinigameColor(((p % 9) % 6 <
3) ^ (p % 54 < 27) ? 0.5 : 0);
                                  }

                                  invs = [0, 0,
0].map((x) => {
                                      let b =
blocks[Math.floor(Math.random() *
blocks.length)];
                                      let s =
Math.random() * 3;
                                      for (let a = 0; a <
s; a++) b = spinInv(b);
                                      b.size =
getInvSize(b);
                                    return b;
                                });
                                for (let b = 0; b < 3;
b++) {
                                    for (let t of
invGrids[b].children)
                                        t.opacity =
invs[b][t.column] ? (invs[b][t.column]
[t.row] == "#" ? 1 : 0) : 0;
                                }

                            }
                        }),
                    ]
                }),
                invGhost = ui.createGrid({
                    margin: new Thickness(0,
0, 0, 6),
                    inputTransparent: true,
                    columnDefinitions: ["1*",
"1*", "1*", "1*"],
                    rowDefinitions: ["1*", "1*",
"1*", "1*"],
                    columnSpacing: 8,
                    rowSpacing: 8,
                    horizontalOptions:
LayoutOptions.START,
                    verticalOptions:
LayoutOptions.START,
                }),
            ]
        })
    });

    popup.show();
}

////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////
///////
```

```
// Domino Puzzle

var floodCount = (pos) => {
   let size =
Math.floor(Math.sqrt(board.length));
   let ans =
Array(board.length).fill(false);
   ans.count = 0;
   let data = +board[pos];
   let stack = [pos];

   while (stack.length > 0) {
      if (board[stack[0]] == data && !
ans[stack[0]]) {
         ans.count++;
         ans[stack[0]] = true;
         if (stack[0] % size < size - 1)
stack.push(stack[0] + 1);
         if (stack[0] % size > 0)
stack.push(stack[0] - 1);
         if (stack[0] < board.length -
size) stack.push(stack[0] + size);
         if (stack[0] >=
size)stack.push(stack[0] - size);
      }
      stack.shift();
   }

   return ans;
}

var showDominoPuzzlePopup = () => {
   if (gameGrid) return;

   board = Array(36).fill("");

   gameGrid = ui.createGrid({
      columnSpacing: 0,
      rowSpacing: 0,
   });
   let c = [];
```

```
    for (let x = 0; x < 6; x++) for (let y =
0; y < 6; y++) c.push(ui.createFrame({
        column: x,
        row: y,
        backgroundColor:
getMinigameColor(0),
        borderColor:
Color.MINIGAME_TILE_BORDER,
        cornerRadius: 4,
        children: [ui.createLatexLabel({
            fontSize: 21,
            horizontalOptions:
LayoutOptions.CENTER,
            verticalOptions:
LayoutOptions.CENTER,
        })],
    }));
    gameGrid.children = c;

    invGrids = [ui.createGrid(),
ui.createGrid(), ui.createGrid()];
    invs = [0, 0, 0].map((x) => {
        let b;
        if (Math.random() < 0.5) b =
[[Math.floor(Math.random() * 6 + 1),
Math.floor(Math.random() * 6 + 1)]];
        else b =
[[Math.floor(Math.random() * 6 + 1)],
[Math.floor(Math.random() * 6 + 1)]];
        b.size = getInvSize(b);
        return b;
    });
    for (let a = 0; a < 3; a++) {
        invGrids[a].columnDefinitions =
invGrids[a].rowDefinitions = ["1*",
"1*"];
        invGrids[a].columnSpacing =
invGrids[a].rowSpacing = 0;
        invGrids[a].column = a;
        let t;
        let c = [];
        for (let x = 0; x < 2; x++) for (let y
= 0; y < 2; y++)
c.push(ui.createFrame({
            column: x,
            row: y,
            heightRequest: 50,
            borderColor:
Color.MINIGAME_TILE_BORDER,
            backgroundColor:
Color.MINIGAME_TILE_LIGHT,
            cornerRadius: 4,
            opacity: invs[a][x] ? (invs[a][x]
[y] ? 1 : 0) : 0,
            children: [ui.createLatexLabel({
                text: invs[a][x] ? (invs[a][x]
[y] ? invs[a][x][y].toString() : "") : "",
                fontSize: 18,
                horizontalOptions:
LayoutOptions.CENTER,
                verticalOptions:
LayoutOptions.CENTER,
            })],
        }));
        invGrids[a].onTouched = (e) => {
            if (invSel && invSel !=
invGrids[a]) return;
            if (!invs[a]) return;

            let t = invGrid.width / 6;
            let pos = [
                e.absoluteX + invGrids[a].x -
t / 2 * (invs[a].size[0] + 1),
                e.absoluteY + invGrid.y - t *
(invs[a].size[1] + 2),
            ];
            let tPos = [
                Math.round(pos[0] / t),
                Math.round((pos[1] -
gameGrid.y) / t),
            ];
            if (e.type.isReleased()) {
                invSel = null;
                let arr =
Array.from(invGhost.children);
                invGhost.children = [];
                invGrids[a].children = arr;

                if (canPlaceInv(invs[a],
tPos)) {
                for (let x = 0; x <=
invs[a].size[0]; x++) for (let y = 0; y <=
invs[a].size[1]; y++) {
                    let p = (tPos[0] + x) * 6
+ tPos[1] + y;
                    if (invs[a][x][y]) {
                        board[p] = invs[a][x]
[y];

gameGrid.children[p].backgroundColo
r = getMinigameColor(1);

gameGrid.children[p].children[0].text
= invs[a][x][y].toString();
                        gameScore += invs[a]
[x][y];
                    }
                }

                invs[a] = null;
                for (let b of
invGrids[a].children) b.opacity = 0;

                if (invs[0] == null &&
invs[1] == null && invs[2] == null) {
                    invs = [0, 0, 0].map((x)
=> {
                        let b;
                        if (Math.random() <
0.5) b = [[Math.floor(Math.random() *
6 + 1), Math.floor(Math.random() * 6 +
1)]];
```

```
                                else b =
[[Math.floor(Math.random() * 6 + 1)],
[Math.floor(Math.random() * 6 + 1)]];
                                b.size =
getInvSize(b);
                                return b;
                            });
                            for (let b = 0; b < 3; b+
+) {
                                for (let t of
invGrids[b].children) {
                                    t.opacity = invs[b]
[t.column] ? (invs[b][t.column][t.row] ?
1 : 0) : 0;
                                    t.children[0].text =
invs[b][t.column] ? (invs[b][t.column]
[t.row] ? invs[b][t.column]
[t.row].toString() : "") : "";
                                }
                            }
                        }

                        let del =
Array(36).fill(false);
                        let delCheck =
Array(36).fill(false);
                        let delNum = 0;

                        for (let p = 0; p < 36; p++)
{
                            if (delCheck[p])
continue;
                            let d = floodCount(p);
                            if (board[p] != "" &&
d.count >= 3) {
                                delNum += d.count;
                                d.map((x, i) => {
                                    if (x) { del[i] =
true; }
                                    return x;
                                });
                            }
                            d.map((x, i) => {
                                if (x) { delCheck[i] =
true; }
                            });
                        }

                        if (delNum > 0) {
                            for (let p = 0; p < 36; p+
+) {
                                if (del[p]) {
                                    board[p] = "";

gameGrid.children[p].backgroundColo
r = getMinigameColor(0);

gameGrid.children[p].children[0].text
= "";
                                }
                            }
                            gameStreak++;
                            gameScore += 5 *
(delNum - 2) * gameStreak;
                        } else gameStreak = 0;

                        if (!canContinue()) {
                            gameGrid.opacity =
0.75;
                            invGrid.opacity = 0;
                            invGrid.heightRequest
-= 50;

endButton.heightRequest = 50;
                            endButton.text = "Gain "
+ Math.floor(gameScore ** 1.1) + " \"↑
time by 0.1s\" clicks";

                            hiscores[1] =
Math.max(hiscores[1], gameScore);
                            hiscoreLabel.text =
hiscores[1].toString();
```

```
                }

                scoreLabel.text =
gameScore.toString();
            }
        } else if (e.type ==
TouchType.PRESSED ||  e.type ==
TouchType.MOVED) {
            if (!invSel) {
                let arr =
Array.from(invGrids[a].children);
                invGrids[a].children = [];
                invGhost.children = arr;
                invGhost.widthRequest =
invGhost.heightRequest =
invGrid.width / 6 * 2 - 16;
            }

            invSel = invGrids[a];
            invGhost.translationX =
pos[0];
            invGhost.translationY =
pos[1];
        }
    }
    invGrids[a].children = c;
};

let popup = ui.createPopup({
    title: "Domino Puzzle",
    onDisappearing: () => {
        clicker.level +=
Math.floor(gameScore ** 1.1);
        hiscores[1] =
Math.max(hiscores[1], gameScore);
        updateScoreEffs();
        gameScore = 0;
        board = gameGrid = invs =
invGrid = invGrids = invSel =
scoreLabel = hiscoreLabel = null;
    },
    content: ui.createGrid({
        children: [
            ui.createStackLayout({
                children: [

ui.createLatexLabel({ text: "Place
dominoes to connect 3 of the same.",
horizontalTextAlignment:
TextAlignment.CENTER, margin: new
Thickness(0, 10, 0, 6) }),
                    ui.createGrid({
                        margin: new
Thickness(0, 0, 0, 6),
                        columnDefinitions:
["1*", "1*"],
                        rowDefinitions: ["28",
"28"],
                        children: [

ui.createLatexLabel({ text: "Score: ",
horizontalOptions:
LayoutOptions.END, row: 0, column: 0,
fontSize: 17, }),
                            scoreLabel =
ui.createLatexLabel({ text: "0", row: 0,
column: 1, fontSize: 17, }),

ui.createLatexLabel({ text: "Best
Score: ", horizontalOptions:
LayoutOptions.END, row: 1, column: 0,
fontSize: 17, }),
                            hiscoreLabel =
ui.createLatexLabel({ text:
hiscores[1].toString(), row: 1, column:
1, fontSize: 17, }),
                        ],
                    }),
                    gameGrid,
                    invGrid = ui.createGrid({
                        margin: new
Thickness(0, 6, 0, 0),
                    columnDefinitions:
["1*", "1*", "1*"],
                    children: invGrids,
                 }),
                 endButton =
ui.createButton({
                    text: "Replay",
                    heightRequest: 0,
                    onClicked: () => {
                       clicker.level +=
Math.floor(gameScore ** 1.1);
                       gameScore = 0;
                       scoreLabel.text =
gameScore.toString();

                       gameGrid.opacity =
1;
                       invGrid.opacity = 1;

invGrid.heightRequest += 50;

endButton.heightRequest = 0;

                       for (let p = 0; p <
36; p++) {
                          board[p] = false;

gameGrid.children[p].backgroundColo
r = getMinigameColor(0);

gameGrid.children[p].children[0].text
= "";
                       }

                       invs = [0, 0,
0].map((x) => {
                          let b;
                          if
(Math.random() < 0.5) b =
[[Math.floor(Math.random() * 6 + 1),
Math.floor(Math.random() * 6 + 1)]];
```

```
                                    else b =
[[Math.floor(Math.random() * 6 + 1)],
[Math.floor(Math.random() * 6 + 1)]];
                                    b.size =
getInvSize(b);
                                    return b;
                                });
                                for (let b = 0; b < 3;
b++) {
                                    for (let t of
invGrids[b].children) {
                                        t.opacity =
invs[b][t.column] ? (invs[b][t.column]
[t.row] ? 1 : 0) : 0;

t.children[0].text = invs[b][t.column] ?
(invs[b][t.column][t.row] ? invs[b]
[t.column][t.row].toString() : "") : "";
                                    }
                                }
                            }
                        }),
                    ]
                }),
                invGhost = ui.createGrid({
                    margin: new Thickness(0,
0, 0, 6),
                    inputTransparent: true,
                    columnDefinitions: ["1*",
"1*"],
                    rowDefinitions: ["1*", "1*"],
                    columnSpacing: 16,
                    rowSpacing: 16,
                    horizontalOptions:
LayoutOptions.START,
                    verticalOptions:
LayoutOptions.START,
                }),
            ]
        })
    });
    popup.show();
}

////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////
///////
// Dice Mastery

var showDiceMasteryPopup = (id) => {
    if (isPopupOpen) return;

    let type = ["d4", "d6", "d8", "d10",
"d12", "d20"][id];

    let cost = new
ExponentialCost(BigNumber.TEN.pow(
[10, 12, 18, 30, 50, 80][id]),
Math.log2(1e5) + Math.log2(100) * id);

    let masters = ["Multiplier",
"Exploding", "Drop Low", "Second
Die", "Critical Level", "Contributor"];
    let masterDesc = [
        "Multiplies roll value by 10",
        "Re-rolls when the highest
number is rolled, results are
compounding, can stack",
        "Multiplies roll value by (dice
number)² when 1 is rolled",
        "Rolls a second die, its result
multiplying roll value",
        "20% chance to multiply roll value
by (dice level)",
        "The base roll value (including
masteries, excluding level multiplier) is
added to a second value",
    ];

    let masterButtons = [];
    for (let a = 0; a < 6; a++) {
```

```
masterButtons.push(ui.createButton({
            text: masters[a], cornerRadius:
0, heightRequest: 50, margin: new
Thickness(0), padding: new
Thickness(10, 0),
            fontFamily:
FontFamily.CMU_REGULAR,
            opacity: diceMasteries[id * 6 +
a] != "#" ? 1 : 0.5,
            inputTransparent:
diceMasteries[id * 6 + a] != "#" ?
false : true,
            onClicked: () => {
                let upg = [d4Master,
d6Master, d8Master, d10Master,
d12Master, d20Master][id];
                let c =
cost.getCost(upg.level);
                if (currency3.value >= c) {
                    let pos = id * 6 + a;
                    if (diceMasteries[pos] !=
"#") {
                        diceMasteries =
diceMasteries.slice(0, pos) + "#" +
diceMasteries.slice(pos + 1);
                        currency3.value -= c;
                        popup.hide();
                        tSkill = id;
                        upg.level += 1;
                    }
                }
            }
        }))
    }

    let popup = ui.createPopup({
        onDisappearing: () =>
isPopupOpen = false,

        content: ui.createStackLayout({
            children: [
                ui.createLatexLabel({ text: "\
\textbf{$" + type + "$ Mastery}",
fontSize: 17, horizontalTextAlignment:
TextAlignment.CENTER }),
                ui.createLatexLabel({
                    text: "Mastery cost: " +
cost.getCost([d4Master, d6Master,
d8Master, d10Master, d12Master,
d20Master][id].level) + "$\\rho_3$",
                    horizontalTextAlignment:
TextAlignment.CENTER, margin: new
Thickness(0, 5, 0, 0)
                }),

ui.createBox({ heightRequest: 1,
margin: new Thickness(0, 8, 0, 0) }),
                ui.createGrid({
                    columnDefinitions: ["1*",
"24", "1*"],
                    margin: new Thickness(0,
5),
                    children: [

ui.createLatexLabel({ text:
"Masteries", verticalTextAlignment:
TextAlignment.CENTER }),
                        ui.createImage({ source:
ImageSource.INFO, heightRequest: 21,
widthRequest: 21, opacity: 0.5,
column: 1, onTouched: (e) => {
                            for (let a = 0; a < 6;
a++) {

masterButtons[a].text =
e.type.isReleased() ? masters[a] :
masterDesc[a];
                            }
                        }}),
                    ],
                }),
                ...masterButtons,

ui.createBox({ heightRequest: 1,
margin: new Thickness(0, 0, 0, 0) }),
                ui.createButton({
                    text:
Localization.get("StarPopupRespec"),
                    onClicked: () => {
                        let upg = [d4Master,
d6Master, d8Master, d10Master,
d12Master, d20Master][id];
                        while (upg.level > 0) {
                            let c =
cost.getCost(upg.level-1);
                            currency3.value += c;
                            upg.level--;
                        }
                        diceMasteries =
diceMasteries.slice(0, id * 6) + "......" +
diceMasteries.slice(id * 6 + 6);
                        popup.hide();
                    }
                })
            ]
        }),
    });

    popup.show();
    isPopupOpen = true;
}

////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////
///////
// Perks and Boosters

var showBoosterPopup = () => {
    if (isPopupOpen) return;

    let costFunc = (x) =>
BigNumber.TEN.pow(80 + x *
20).pow(x) *
BigNumber.from("e1000");
    let cost =
costFunc(pBoostersBought);

    let sum = 0;
    let ppText, ppButton,
pBoosterButtons = [];

    let types = ["Data Collection",
"Analyzation", "Evaluation",
"Revision"];

    for (let a = 0; a < 4; a++) {
        pBoosterButtons[a] =
ui.createButton({
            heightRequest: 60,
            cornerRadius: 0,
            onClicked: () => {
                if (sum < pBoostersBought) {
                    pBoosters[a]++;
                    updateButtons();
                }
            },
        });
    }

    let updateButtons = () => {
        sum = pBoosters.reduce((x, y) =>
x + y);
        if (ppText) ppText.text = "You
have " + sum + " / " +
pBoostersBought + " boosters.";

        for (let a = 0; a < 4; a++) {
            pBoosterButtons[a].text =
types[a] + " - " + pBoosters[a] + "
allocated\n×" + (1 + pBoosters[a] *
(perks[31 + a] ? 2 : 1)).toFixed(1) + "
ATK boost";
            pBoosterButtons[a].opacity =
sum < pBoostersBought ? 1 : 0.5;

pBoosterButtons[a].inputTransparent
= sum < pBoostersBought ? false :
true;
        }
    }
    updateButtons();

    let popup = ui.createPopup({
        isPeekable: true,
        onDisappearing: () =>
isPopupOpen = false,

        content: ui.createStackLayout({
            children: [
                ui.createLatexLabel({ text: "\
\textbf{\\textsf{ATK} Boosters}",
fontSize: 17, horizontalTextAlignment:
TextAlignment.CENTER }),
                ppText =
ui.createLatexLabel({
                    horizontalTextAlignment:
TextAlignment.CENTER,
                    text: "You have " + sum +
" / " + pBoostersBought + " boosters.",
                    margin: new Thickness(0,
15, 0, 5),
                }),
                ppButton = ui.createButton({
                    text: "Get 1 at " + cost + "
publication multiplier",
                    opacity: () =>
theory.publicationMultiplier >= cost ?
1 : 0.5,
                    inputTransparent: () =>
theory.publicationMultiplier >= cost ?
false : true,
                    onClicked: () => {
                        if
(theory.publicationMultiplier >= cost) {
```

```
                            pBoostersBought++;
                            cost =
costFunc(pBoostersBought);
                            ppButton.text = "Get
1 at " + cost + " publication multiplier";
                            updateButtons();
                        }
                    },
                }),

ui.createBox({ heightRequest: 1,
margin: new Thickness(0, 8) }),
                ...pBoosterButtons,

ui.createBox({ heightRequest: 1,
margin: new Thickness(0, 8) }),
                ui.createButton({
                    text:
Localization.get("StarPopupRespec"),
                    onClicked: () => {
                        pBoosters = [0, 0, 0, 0];
                        updateButtons();
                    }
                }),
                ui.createButton({ text:
Localization.get("AutoPrestigeClose"),
onClicked: () => { popup.hide() } })
            ]
        }),
    });

    popup.show();
    isPopupOpen = true;
}

var showPerkPopup = () => {
    if (isPopupOpen) return;

    let lines = {};
    let buttons = {};
    let isSubPopupOpen = false;

    let costFunc = (x) =>
BigNumber.TEN.pow(8 + x * 2).pow(x)
* 1e160;
    let cost =
costFunc(perkPointsBought);

    let pass = (p) => {
        let data = perkData[p];
        if (data.req) for (let req of
data.req) if (!perks[req]) {
            return false;
        };
        return true;
    }

    for (let p in perkData) {
        let data = perkData[p];
        let line = Math.floor(p / 10);
        if (!lines[line]) {
            lines[line] =
ui.createStackLayout({
                orientation:
StackOrientation.HORIZONTAL,
                horizontalOptions:
LayoutOptions.CENTER,
                children: [],
            })
        }
        lines[line].children =
[...lines[line].children, buttons[p] =
ui.createButton({
            text: p,
            heightRequest: 48,
            widthRequest: 48,
            cornerRadius: 0,
            onClicked: () => {
                if (isSubPopupOpen) return;

                let buyButton =
ui.createButton({
                    onClicked: () => {
                        if (perkPoints >= 1 && !
perks[p] && pass(p)) {
                            perks[p] = true;
                            perkPoints--;
                            buyButton.text =
Localization.get("BuyablesCostBought
");
                            buyButton.opacity =
0.5;

buyButton.inputTransparent = true;
                            updateButtons();
                            updateAvailability();

theory.invalidateSecondaryEquation();
                        }
                    }
                });

                buyButton.text = perks[p] ?
Localization.get("BuyablesCostBought
") : pass(p) ? "1 Perk Point" : "Requires
" + data.req.map(x =>
perkData[x].name).join(" + ");
                buyButton.opacity = !
perks[p] && pass(p) && perkPoints >=
1 ? 1 : 0.5;
                buyButton.inputTransparent
= !perks[p] && pass(p) && perkPoints
>= 1 ? false : true;

                let subPopup =
ui.createPopup({
                    onDisappearing: () =>
isSubPopupOpen = false,
                    title: data.name,
                    content:
ui.createStackLayout({
                        children: [
                            ui.createLatexLabel({

horizontalTextAlignment:
TextAlignment.CENTER,
                                text: data.info,
                                margin: new
Thickness(0, 10, 0, 10),
                            }),
                            buyButton,

ui.createBox({ heightRequest: 1,
margin: new Thickness(0, 8) }),
                            ui.createButton({ text:
Localization.get("AutoPrestigeClose"),
onClicked: () => { subPopup.hide() } })
                        ]
                    }),
                });

                subPopup.show();
                isSubPopupOpen = true;
            },
        })];
    }

    let ppText, ppButton;

    let updateButtons = () => {
        if (ppText) ppText.text = "You
have " + perkPoints + " perk points.";
        for (let p in buttons) {
            buttons[p].backgroundColor =
perks[p] ?
Color.MINIGAME_TILE_LIGHT :
Color.MEDIUM_BACKGROUND;
            buttons[p].opacity = pass(p) ?
1 : 0.5;
        }
    }
    updateButtons();
    let popup = ui.createPopup({
        isPeekable: true,
        onDisappearing: () => isPopupOpen = false,
        title: "Perk Panel",

        content: ui.createStackLayout({
            children: [
                ppText = ui.createLatexLabel({
                    horizontalTextAlignment: TextAlignment.CENTER,
                    text: "You have " + perkPoints + " perk points.",
                    margin: new Thickness(0, 5, 0, 5),
                }),
                ppButton = ui.createButton({
                    text: perkPointsBought < 11 ? "Buy 1 for " + cost + " ρ₃" : "All perk points bought!",
                    opacity: () => currency3.value >= cost && perkPointsBought < 11 ? 1 : 0.5,
                    inputTransparent: () => currency3.value >= cost && perkPointsBought < 11 ? false : true,
                    onClicked: () => {
                        if (currency3.value >= cost && perkPointsBought < 11) {
                            currency3.value -= cost;
                            perkPoints++;
                            perkPointsBought++;
                            cost = costFunc(perkPointsBought);
                            ppText.text = "You have " + perkPoints + " perk points.";
                            ppButton.text = perkPointsBought < 11 ? "Buy 1 for " + cost + " ρ₃" : "All perk points bought!";
                            updateAvailability();

    theory.invalidateSecondaryEquation();
                        }
                    }
                }),

    ui.createBox({ heightRequest: 1,
    margin: new Thickness(0, 8) }),
                ui.createScrollView({
                    content:
    ui.createStackLayout({
                        children:
    Object.values(lines),
                    }),
                }),

    ui.createBox({ heightRequest: 1,
    margin: new Thickness(0, 8) }),
                ui.createButton({
                    text:
    Localization.get("StarPopupRespec"),
                    onClicked: () => {
                        perks = {};
                        perkPoints =
    perkPointsBought;
                        updateButtons();
                        updateAvailability();

    theory.invalidateSecondaryEquation();
                    }
                }),
                ui.createButton({ text:
    Localization.get("AutoPrestigeClose"),
    onClicked: () => { popup.hide() } })
            ]
        }),
    });

    popup.show();
    isPopupOpen = true;
```

```
}

init();
```